Prototype Tool for Supporting a Formal Engineering Approach to Service-Based Software Modeling

Weikai Miao[1(✉)] and Shaoying Liu[2]

[1] Shanghai Key Laboratory of Trustworthy Computing,
Software Engineering Institute, East China Normal University, Shanghai, China
wkmiao@sei.ecnu.edu.cn
[2] Department of Computer Science, Hosei University, Tokyo, Japan
sliu@hosei.ac.jp

Abstract. Despite the advances in service-based software modeling, few existing approaches and tools support a systematic engineering process in which precise specification construction and accurate web service selection are integrated coherently. Due to this reality, how to carry out service-based software modeling so that existing services can be accurately discovered, selected, and effectively reused in the system under development is still a challenge. To solve this problem, this paper describes a prototype tool that supports a formal engineering framework for service-based software modeling. Formal specification can be constructed in an evolutionary manner; meanwhile, appropriate services are discovered and selected through the specification evolution. We illustrate the basic principle underlying the tool. The tool design and its implementation are also described. An example is presented to demonstrate major features of this tool.

1 Introduction

Research on engineering methods for constructing high quality service-based software (*service-based software*) is attracting a growing attention of both research and industry communities in recent years [13,16]. Constructing formal specifications, including both formal requirements and design specifications, based on correct understanding of requirements contributes significantly to software quality [15]. To effectively support the service-based software modeling, this fundamental principle needs to be extended.

One major problem with existing service-based software modeling is how to carry out the modeling so that existing services can be accurately discovered, selected, and effectively reused in the system under development. To tackle this challenge, effective engineering methods are demanded. An effective engineering method for service-based software modeling needs to supply definitive mechanisms for eliciting requirements, constructing precise specifications, selecting appropriate services, and integrating these artifacts coherently into a system model.

S. Liu and Z. Duan (Eds.): SOFL+MSVL 2013, LNCS 8332, pp. 89–103, 2014.
DOI: 10.1007/978-3-319-04915-1_7,

Many research efforts have been advanced service-based software modeling from different perspectives, including business process modeling and implementation [1,2,5,18] and service discovery and selection [6,11]. Unfortunately, as summarized by the authors of work [16], almost no approach supports service discovery and selection as part of system modeling. That is, service discovery and selection activities are not coherently integrated into the system modeling phase. Therefore, services may not be effectively and efficiently adopted in the system architecture.

To tackle the above challenge, we have proposed a new approach called *Formal Engineering Framework for Service-based Software Modeling* (*FEFSSM*) as a solution [12]. FEFSSM integrates service discovery and selection into the entire modeling procedure, aiming to provide a unified engineering approach to constructing precise, comprehensible, and satisfactory specifications of service-based software.

To facilitate the application of FEFSSM in practice, in this paper we describe a supporting tool of the FEFSSM approach. The tool implements the fundamental principle underlying the FEFSSM, offering basic functions that support the involved engineering activities. The practitioner can construct the formal specification gradually through the interaction with the tool.

The rest of this paper is organized as follows. Basic theory of the FEFSSM approach underlying the tool is presented in Sect. 2. Design and implementation of the tool is described in Sect. 3. Section 4 describes an example of modeling a travel agency system to demonstrate the usability of the tool. Section 5 gives the comparison with the related work. Finally, we conclude the paper and point out future research directions in Sect. 6.

2 The FEFSSM Approach

The principle of FEFSSM inherits from the well-established SOFL (Structured Object-oriented Formal Language) formal engineering method [7,9,14,15] but emphasizes the interleaving and interaction of *software modeling* and *service adoption* in building service-based software. The main principle of FEFSSM is illustrated in Fig. 1.

Web service discovery, filtering, and selection activities are carried out sequentially in coupling with the corresponding specification construction activities in a three-step modeling process through the informal, semi-formal and formal stages of specification construction. The rational and technical details of each step are described below.

(1) **Informal specification construction**

The goal of informal specification construction is to acquire requirements as completely as possible at an informal level and discover sufficient candidate services based on the informal requirements. Requirements acquisition is usually achieved through communication between the client and the developer. Since requirements are imprecise at this stage, candidate services are preliminarily explored and filtered using keywords that abstract the corresponding informal

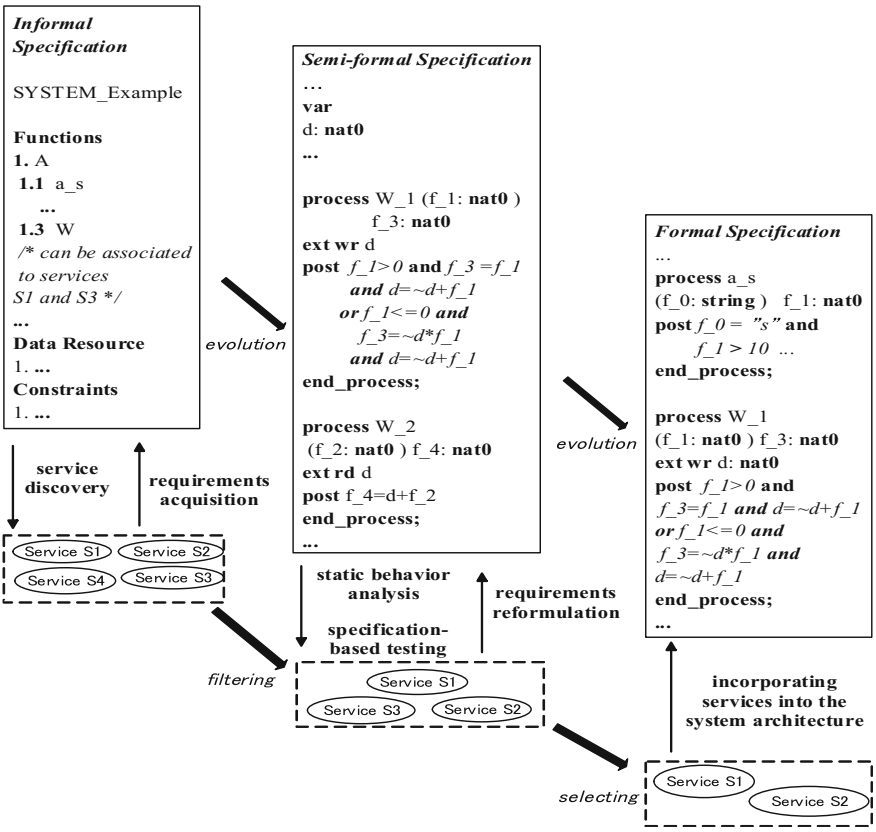

Fig. 1. The FEFSSM approach

requirements. The derived keywords are used to either partially or completely match with the service names or informal descriptions stored in the service repositories. A precise criterion for associating a service as a candidate service to an expected function is defined below.

Criterion 1. *Let $K_r = \{a_1, a_2, ..., a_n\}$ be a set of keywords derived from function F in the informal specification and $K_s = \{b_1, b_2, ..., b_m\}$ be a set of keywords derived from the name and informal descriptions of a service S. Then, S is accepted as a candidate service to be associated with F if and only if the following condition holds:*

$$\exists_{a \in K_r} \exists_{b \in K_s} \cdot is\_substring(a, b)$$

This criterion states that if there exists a keyword derived from a function F in the specification that is a substring (case insensitive) of a keyword derived from the name and informal descriptions of a service S, S is accepted as a candidate service to be associated with F.

In FEFSSM, service searching is in parallel with the functional decomposition. For example, since the developer does not find any candidate service for function A, then A is decomposed into several sub-functions. For each sub-function, the developer tries to explore candidate services. As the result, the sub-function W is associated to two candidate services *S1* and *S3*. Detailed algorithm that encompasses the procedure of service discovery and functional decomposition is proposed in our previous work [12].

The ultimate informal specification contains three sections: *functions*, *data resources*, and *constraints*. The *functions* section briefly describes the desired functions of the target system, which are usually organized in a hierarchical structure. The *data resources* section presents the necessary data items for building the system function. The *constraints* section documents the required constraints on either the functions or the data resources.

(2) **Semi-formal specification construction**

The purpose of semi-formal specification construction is to evolve the informal specification into a more precise, complete, and well structured specification that encompasses accurately selecting services as some of its functional components; meanwhile, the services can be used for reformulating the specification. In regard to service selection, services are accurately selected through static behavior analysis and specification-based conformance testing.

– Static behavior analysis

The essential idea of the static behavior analysis is to judge which candidate service is most suitable for implementing each service-associated function. Specifically, the developer extracts potential functions of the services by analyzing their descriptions files (e.g., *WSDL* file) and finally identify the most relevant services for each service-associated function. The relevance of each candidate service to its associated function is represented by a ranking score. The service with the highest ranking score is selected as the most relevant service of its associated function. The ranking procedure can be referred to the corresponding algorithm in our work [12].

As pointed out by Guideline 1 [12], if function F is associated to its most relevant service S, then F is refined into a set of sub-functions $\{f_1, ..., f_n\}$ $(n \geq 1)$ where each sub-function f_i is associated to the corresponding operation provided by S. These sub-functions are then transformed into formal *processes* that specify the intended functions precisely. A formal process is written in the SOFL specification language [14], which includes a signature, a pre-condition and a post-condition. The signature includes the input, output, and external variables (or state variables); the pre-condition imposes a constraint on the input variables before executing the process; and the post-condition describes a condition that must be satisfied by the output and external variables after the execution of the process.

After analyzing the descriptions of two candidate services *S1* and *S3* of function *W*, the developer identifies *S1* as the most relevant service. Then function *W* is refined into two sub-functions *W_1* and *W_2*. These two sub-functions are further formalized as formal processes *W_1* and *W_2*.

– Specification-based testing
The specification-based conformance testing is aimed at dynamically checking whether a service satisfies the required functions defined by the corresponding formal processes. Test cases are generated from the pre- and post-conditions of the processes and the final decision to accept or reject the service is made by the developer based upon test results analysis and engineering judgements. To facilitate a rigorous testing, each process is converted into an equivalent disjunction of *functional scenarios*, each describing an independent function in terms of the input and output relation [8].

Definition 1. *Let P denote a process and its* post-condition $P_{post} \equiv (C_1 \wedge D_1) \vee (C_2 \wedge D_2) \vee ... \vee (C_n \wedge D_n)$*, where each C_i $(i = 1, ..., n)$ is a predicate called a* guard condition *that contains no output variable and D_i a* defining condition *that contains at least one output variable but no* guard condition*. Then, each $P_{pre} \wedge C_i \wedge D_i$ is called a* functional scenario*.*

Functional scenarios are used as the foundation of test data derivation and also the test oracles. One intuitive way to test each service operation is to generate test cases that cover every functional scenario of the associated formal process. To test stateless service operations (i.e. execution results are determined by only the input values), test cases are directly derived from single functional scenarios. To sufficiently test stateful service operations (i.e. executions results are determined by both the input values and internal stateful variables that cannot be directly monitored from the user-end), all pairs of functional scenarios produced by the inter-related processes (i.e. processes that share the same data stores) are adopted for generating test sequences of the corresponding service operations [12].

When services are determined via the conformance testing, the specification can be transformed into a semi-formal specification. All of the related functions, data resources, and constraints in the informal specification are grouped into SOFL *modules*, each containing declarations of *types*, *state variables*, *invariants*, and *processes*. In each module, all of the declarations of types and variables are expressed formally but the logic-related parts such as invariants and processes are expressed informally to represent the expected functions (except service-associated functions, as explained below).

(3) **Formal specification construction**

The final stage of modeling is to transform the semi-formal specification into a formal design specification. The transformation is achieved by formally defining the system architecture into a hierarchical structure and formalizing the pre- and post-conditions of all the processes. The key point is that service-associated processes in the semi-formal specification are used as the foundation for gradually formalizing the entire specification since they have been determined to be part of the target system at the previous modeling stage.

In the formal specification shown in Fig. 1, all the processes including the service-associated process *W_1* and process *a_s* that is not associated to any service, are formally defined.

3 Design and Implementation of the FEFSSM Tool

3.1 Tool Design

The tool is designed and implemented to facilitate the usability of the FEFSSM. It guides the practitioner to follow the entire engineering process of the FEFSSM approach and support the automation of some specific activities such as the service discovery, service ranking and functional scenario pairs generation. Meanwhile, it also offers appropriate interfaces to handle the interactions between the practitioner and the specification components.

The tool is designed as a three-layered system which provides the major functionalities supporting the application of the FEFSSM. The architecture of this supporting tool is described by Fig. 2.

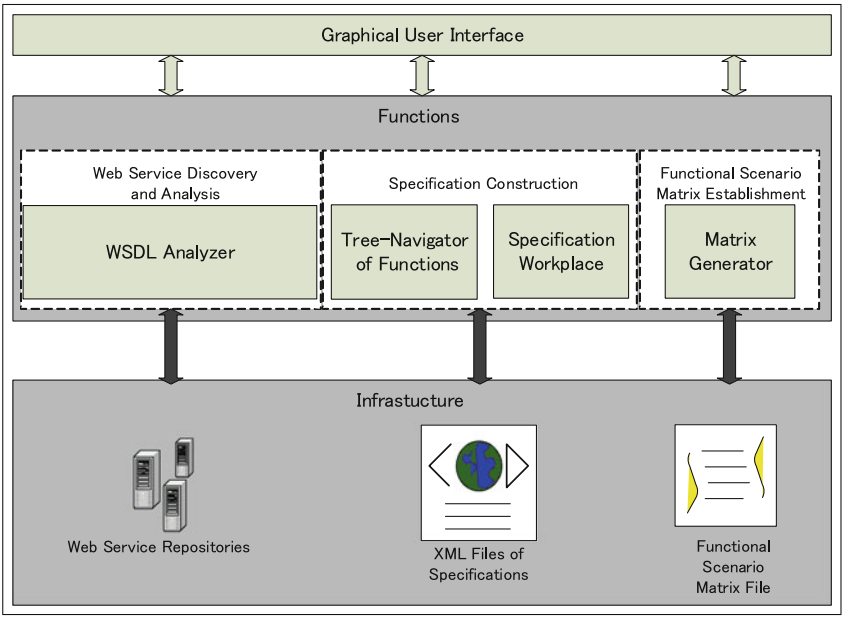

Fig. 2. Architecture of the FEFSSM tool

The infrastructure layer refers to the necessary documents and artifacts that support specification construction and service selection. These artifacts mainly consist of the service repositories information, the specification files and the files of functional scenario matrices. Usually service repositories information documents basic descriptions of the available web services. Specifications are documented in XML files. We use the XML files since the they are machine-readable, platform-independent and can easily represent the hierarchical structures of the

processes and other SOFL components. To store the functional scenario pairs for web service testing, functional scenario pairs are organized as functional scenario matrix that is saved as a functional scenario matrix file. Each row of the matrix is a functional scenario pair that can be read by the tool for further test sequence generation.

The function layer consists of three modules: service discovery and analysis module, specification construction module, and functional scenario matrix module. Service discovery and analysis module supports the keyword-based web service discovery and the static behavior analysis, which is performed by a WSDL Analyzer. The analyzer can extract detailed interface information from the WSDL files of the available web services.

The module of specification construction is responsible for documenting the specifications. Specifically, we provide a graphical tree-navigator as a short-cut for function decomposition and documentation. The practitioner can directly manage the hierarchy structures of expected functions rather than manually typing them. Specification workplace performs the basic functions for editing specifications in different modeling phases.

Functional scenario matrix module contains a matrix generator for constructing the functional scenario matrices for test sequence generation.

3.2 Tool Implementation

The tool is implemented in Java language under the Eclipse environment. Figure 3 gives a screenshot of the main interface of the tool.

The text edition area located in the left-side is the workplace for specification construction. The practitioner can shift the three tabs on the top of the text edition area to edit the specifications in different stages. This screenshot shows the interface of informal specification construction. The tree-navigator is in the middle part, which is labelled as "*hierarchical*". A tree structure of expected functions is described by the navigator in which each node is an expected function. By right-clicking the node, the practitioner can decompose, delete, edit the function or search candidate services. Discovered services are listed in the area labelled as "*Discovered Services of Function*". The right-side of this interface lists all the available services in the service repositories.

Specification Construction. Specification construction is implemented by the specification workplace and the tree-navigator.

Specification workplace offers the basic functions for constructing specifications, focusing on the writing and reading operations on the XML files of the potential informal, semi-formal and formal specifications. Figure 4 shows an example of the informal specification stored as an XML file.

In this XML file, hierarchical structures of the functions are represented by the hierarchy of the XML elements. For example, function *Lowest_1* is the child node of function *Function_1_Child_1* that is the child node of its higher-level function *Function_1*. Associated services of each function are also documented

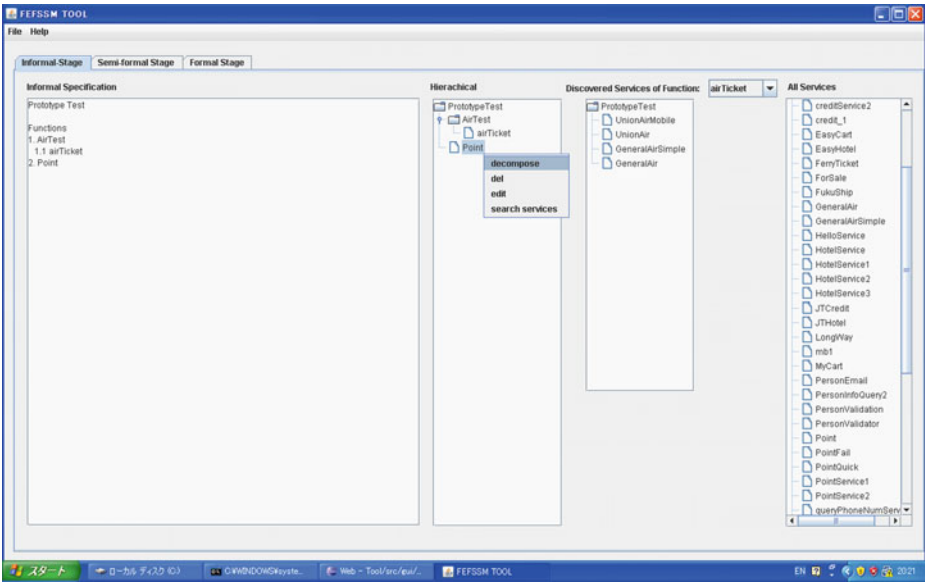

Fig. 3. The screenshot of the main interface of the tool

in the XML file. In this example, after the keyword-based service searching, basic information of a discovered service *JTHotel* is recorded and associated to function *Lowest_1*. As one advantage of the XML format, we can easily locate any element of the specification using *XPath* commands.

The tool provides the practitioner with a tree-navigator to directly decompose, delete or edit the expected functions. Service discovery can also be carried out by this navigator. In the navigator shown in Fig. 3, function *AirTest* and its sub-function *airTicket* and function *Point* are displayed in a consistent hierarchical structure of the textual specification. By right-clicking any function in the navigator, the practitioner can decide to modify the function (i.e. decompose, delete or rename the function) or start the keyword-based service searching for the function.

3.3 Service Discovery and Analysis

Criterion 1 of the keyword-based service discovery is implemented by a matching algorithm. A set of keywords stored in array *key* are splitted into single keywords. Each keyword is then compared with each service name stored in array *allServices*. The discovered services are collected as candidate services that are associated to the specified function in the informal specification. The associations between candidate services and the expected function are added into the XML file of the specification. For example, service *JTHotel* is associated to function *Lowest_1*, which is described in Fig. 4.

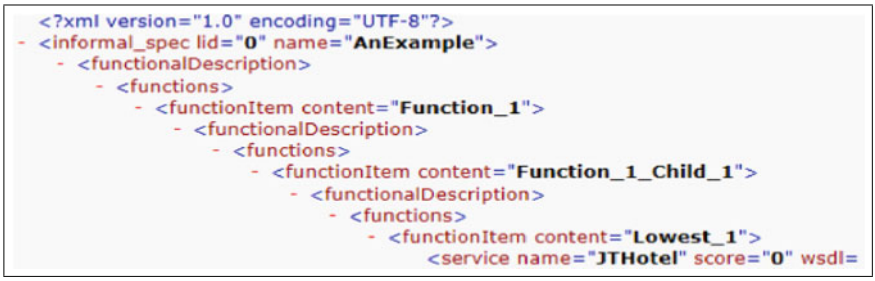

Fig. 4. XML file of informal specification

Following the FEFSSM approach, the practitioner needs to analyze the candidate services based on their interface descriptions to extract the potential functional behaviors and then identify the most relevant services for further conformance testing. The analysis is realized by the Service Analyzer through analyzing the WSDL files of the services. Figure 5 describes the kernel operations of this analyzer.

```
public Opr[] getOperations(String wsdl){
  ...
List oprlist = XPath.selectNodes(doc, "//wsdl:portType//wsdl:operation");
  ...
}

public parameterModel[] singleTypeAnalysis(String analysisName, String wsdl){
 // extract parameter information from WSDL
      ...
List list1 = XPath.selectNodes(doc, "//xs:schema/xs:element[@name='"+analysisName+"']/xs:complexType//xs:element");
      ...
}
```

Fig. 5. Operations for analyzing WSDL file

The interface information, including the available operations and their input/output parameters of each service, is extracted from the WSDL files by two methods *getOperations* and *singleTypeAnalysis*. The tool also offers a graphical interface for dealing with the relevance score ranking, which will be demonstrated in the next section.

3.4 Functional Scenario Matrix Establishment

Functional scenario matrix is constructed by the tool for conformance testing. The code shown in Fig. 6 implements the matrix generation.

For N functional scenarios accepted by method *matrixOperation*, array *result* records the N^2 functional scenario pairs. Each element of this array is a pair of functional scenarios, which is established by the two *for* loop statements.

```
public static String[] matrixOperation(String[] scenarios){
  String[] result = new String[scenarios.length*scenarios.length];
  int index=0;
  for(int i = 0; i < scenarios.length; i++){
   for(int j = 0; j < scenarios.length; j++){
     String temp=(String)scenarios[i]+","+(String)scenarios[j];
       result[index]=temp;
       index++;
             ...
```

Fig. 6. Operation of functional scenario matrix generation

4 An Example

To demonstrate the usability of the tool, we have conducted an example of modeling a *Travel Agency System* (*TAS*) using this prototype tool. Some students in our research group act as the practitioner to model the TAS.

TAS modeling starts from the informal specification construction. The practitioner records the expected functions using the tree-navigator of the tool. In the navigator, the practitioner defines a function *Hotel Operation* and then tries to discover available web services to implement this functions. Meanwhile, corresponding textual informal specification is updated in the specification edition area. A set of keywords is then given by the practitioner to search candidate services. The service discovery procedure is described by Fig. 7.

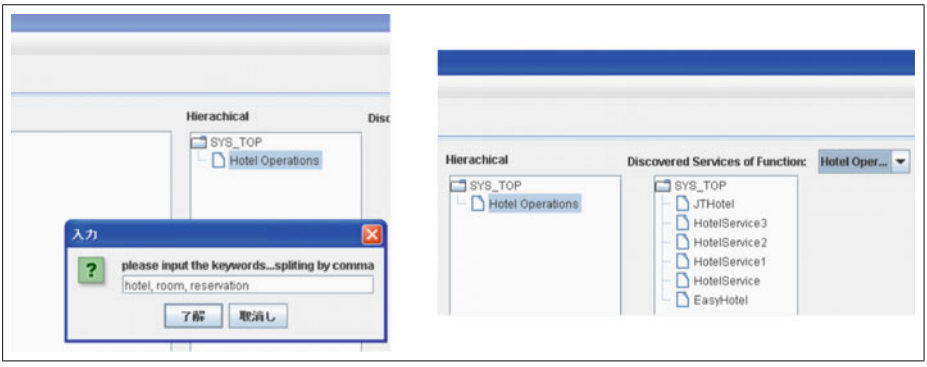

Fig. 7. Keyword-based service discovery

In the left part of Fig. 7, three keywords "*hotel*", "*rooms*" and "*reservation*" are decided. As the result of service searching, six candidate services, for example, service *JTHotel*, are listed in the interface, which is shown by the right part of this figure. Names of all the discovered services match the keyword "*hotel*".

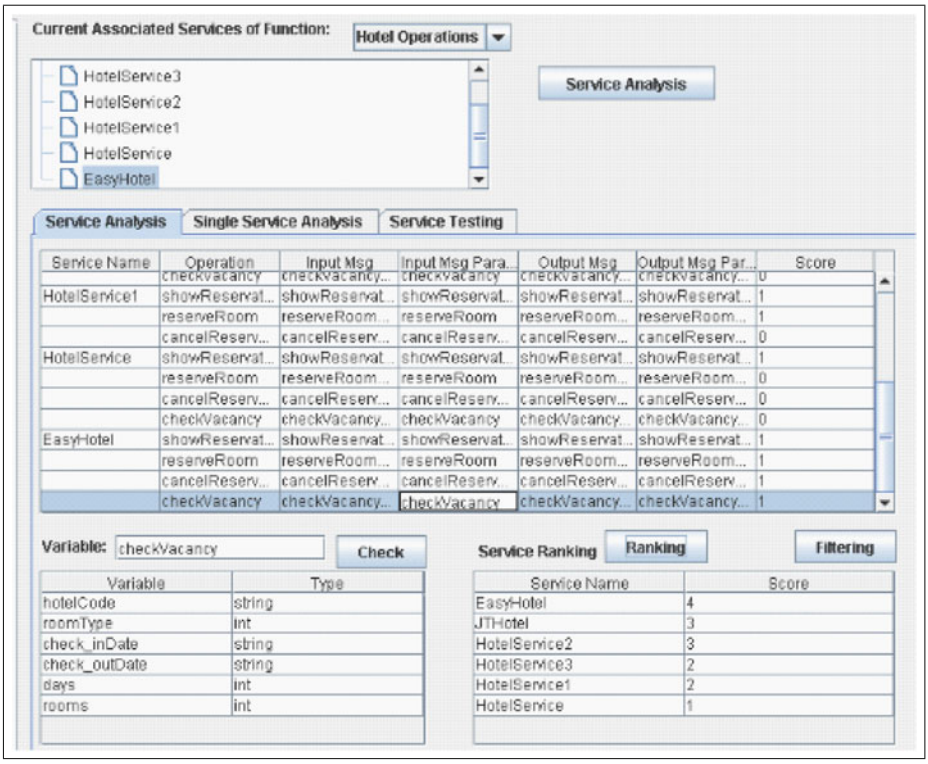

Fig. 8. Static behavior analysis of services

When the informal specification is finished, by shifting the "*semi-formal stage*" tab, the practitioner can start the semi-formal specification construction. The first step of this stage is to carry out the static behavior analysis of the candidate services. Figure 8 shows the interface for static behavior analysis of candidate services.

Static behavior analysis is invoked when the practitioner clicks the button "*Service Analysis*". The interface information, including the service names, operations, and input/output messages of all the candidate services associated to the corresponding function, are listed in a table. For example, service *EasyHotel* has an operation *checkVacancy*. The input message of this operation is also named as *checkVacancy*. The practitioner can select this message and check its detailed variables by clicking the button "*Check*". For instance, six variables (e.g., *hotelCode*, *roomType*, *rooms* and etc.) constitute the message *checkVacancy*. Based on the detailed descriptions of each service operation, the practitioner can judge which operation is necessary for implementing the expected function and then assign the ranking scores for the operations. The practitioner accepts operation *checkVacancy* of service *EasyHotel* after a thorough understanding of its input and output variables, then this operation gets one point as its ranking score.

Similarly, ranking scores of all the services can be assigned and sorted. Finally, these ranking scores are displayed the right-side of this interface. In this case study, the most relevant service is *EasyHotel* since its gets the highest ranking score of four points.

By clicking the button "*Filtering*" on the interface of static behavior analysis, only the most relevant service will be reserved for conformance testing. The tool automatically generates the corresponding framework of SOFL processes that associated to the most relevant service, which is described by Fig. 9.

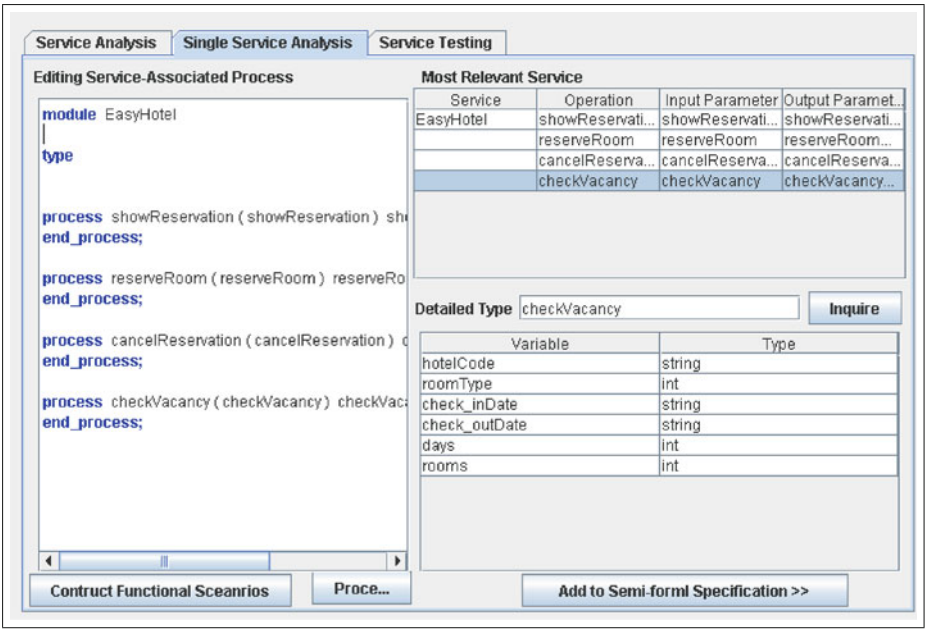

Fig. 9. Formalizing the processes associated to the most relevant service

For the most relevant service *EasyHotel*, the tool automatically generates four processes that correspond to its four operations. The input and output structures are also constructed. For example, process *checkVacancy* which takes variable checkVacancy as its input data is generated. The practitioner can further clarify the data types referring to the corresponding parameters of the service operations. When the data structures of the processes are determined, formal functional scenarios of these processes can be defined. Once all the functional scenarios of the processes are constructed, functional scenario matrix can then be generated. Figure 10 describes the procedure of defining formal functional scenarios of the processes and the generated functional matrix.

Each functional scenario is assigned with an identifier. For instance, functional scenario *f_1* stands for the functional scenario of a successful inquiry of reservation. In our example, for the four processes associated to the candidate

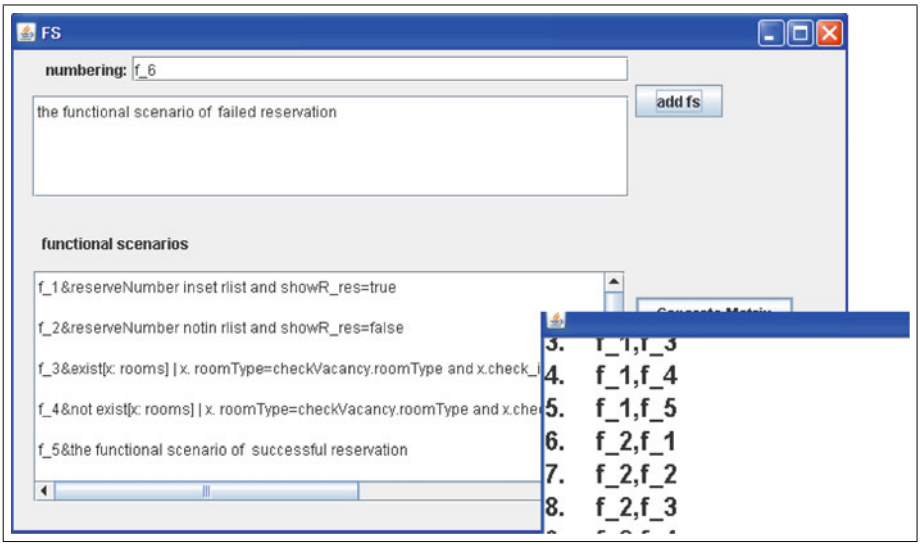

Fig. 10. Interface of functional scenarios construction

service *EasyHotel*, eight functional scenarios are constructed. As the result, a 64*2 functional scenario matrix is generated. For example, the fourth row in the matrix is functional scenario pair (*f_1, f_4*).

Based on the functional scenario matrix, test sequences are derived for the conformance testing of services. Currently, test data generation cannot be fully supported by the prototype tool. Another testing tool developed by our group can be exploited to finish the testing [10]. As an important task of our future research, we will make efforts to integrate the functionalities of the previous testing tool into the FEFSSM tool so that the conformance testing can be more effective and efficient.

Assume the services are determined, the practitioner can proceed to construct the semi-formal and formal specifications, which are supported by the corresponding specification editing areas.

5 Related Work

Various tools have been developed to support service-based software modeling from different perspectives. One category of them focuses on service-based software modeling based on business process modeling techniques (e.g., BPMN). A CASE tool called WebRatio is extended to support model-driven services integration [4]. The extended tool supports the high-level business process modeling of BPMN notations and detailed services mashup application modeling using WebML language. A service-oriented business process modeling tool is proposed in work [3]. It supports the generation of business process model from BPMN-based business process meta-model with highlighted web service characteristic

and automatic translation and deployment features. The authors of work [5] report a prototype tool that provides the functionality of graphically modeling a BPEL process and running static validation. The methodology underlying this tool is using the graphical aspect of BPMN in order to facilitate modeling of executable BPEL service orchestrations. However, since BPMN notation is lack of formal semantics, these BPMN-based modeling methodologies and their supporting tools do not support precise specification construction of service-based software. In the work [18], the authors propose a formal semantics of BPMN defined in terms of a mapping to YAWL nets, for which efficient analysis techniques exists. The proposed mapping has been implemented as a supporting tool. In the work [17], the authors also propose formal semantics of the BPMN notation. These approaches and tools contribute to the service-based software modeling, especially from the perspective of modeling notations while appropriate service-based software modeling methodology is not addressed. Practitioners are demanding engineering methodologies to guide them effectively exploit these techniques.

Moreover, web services selection is not considered by these modeling approaches or techniques. As the authors of work [16] summarize, few approaches support service discovery and selection as part of the design process of service-based systems. Work [16] proposes a method that is relevant to our FEFSSM, in which services discovery and selection are integrated into the entire service-based software modeling process. Specifically, the service discovery in work [16] is carried out through semantics-based matching. Services are not required to be dynamically invoked through the matching process. Service selection in our FEFSSM methodology is realized via formal specification-based conformance testing in which services are dynamically tested after a preliminary keyword-based matching and a static analysis procedures.

6 Conclusion

To facilitate service-based software modeling, we present an interactive tool that supports the FEFSSM approach. We illustrate the theory of FEFSSM underlying the tool and describe its design and implementation. An example of modeling a travel agency system (TAS) is illustrated to demonstrate the usability of the tool.

At present, the tool supports the engineering process and most activities of the FEFSSM while the conformance testing of service selection has not been fully realized. In our future research, we will complete the tool so that the FEFSSM approach can be applied more effectively in practice. We are also interested in the techniques of constructing high-quality service-based software system.

Acknowledgement. This research is supported by SCAT research Foundation. This research is also supported by IDEA4CPS, MT-LAB (VKR Centre of Excellence), Shanghai Knowledge Service Platform for Trustworthy Internet of Things No. ZF1213 and NSFC Project No.91118007.

References

1. http://www.bpmn.org/ (2011) (Online)
2. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf (2011) (Online)
3. Bai, L., Wei, J.: A service-oriented business process modeling methodology and implementation. In: International Conference on Interoperability for Enterprise Software and Applications China, 2009, IESA '09, April 2009, pp. 201–205 (2009)
4. Bozzon, A., Brambilla, M., Facca, F., Carughu, G.: A conceptual modeling approach to business service mashup development. In: IEEE International Conference on Web Services (ICWS), July 2009, pp. 751–758 (2009)
5. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: extending BPEL for modeling choreographies. In: International Conference on Web Services (ICWS), July 2007, pp. 296–303 (2007)
6. Klusch, M., Fries, B., Sycara, K.: Automated semantic web service discovery with OWLS-MX. In: Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Hakodate, Japan, May 2006, pp. 915–922 (2006)
7. Liu, S., Chen, Y., Nagoya, F., McDermid, J.: Formal specification-based inspection for verification of programs. IEEE Trans. Softw. Eng. **99** (2011, PrePrints)
8. Liu, S.: Integrating specification-based review and testing for detecting errors in programs. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 136–150. Springer, Heidelberg (2007)
9. Liu, S., McDermid, J., Chen, Y.: A rigorous method for inspection of model-based formal specifications. IEEE Trans. Reliab. **59**(4), 667–684 (2010)
10. Liu, S., Nakajima, S.: A "Vibration" method for automatically generating test cases based on formal specifications. In: 18th Asia-Pacific Software Engineering Conference (APSEC2011), Ho Chi Minh, Vietnam, December 2011, pp. 73–80 (2011)
11. Meditskos, G., Bassiliades, N.: Structural and role-oriented web service discovery with taxonomies in OWL-S. IEEE Trans. Knowl. Data Eng. **22**(2), 278–290 (2010)
12. Miao, W., Liu, S.: A formal engineering framework for service-based software modeling. IEEE Trans. Serv. Comput. **6**(4), 536–550 (2013)
13. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: state of the art and research challenges. Computer **40**, 38–45 (2007)
14. Liu, S.: Formal Engineering for Industrial Software Development Using the SOFL Method. Springer, Heidelberg (2004)
15. Liu, S., Offutt, A.J., Ho-Stuart, C., Sun, Y., Ohba, M.: SOFL: a formal engineering methodology for industrial applications. IEEE Trans. Softw. Eng. **1**, 24–45 (1998)
16. Spanoudakis, G., Zisman, A.: Discovering services during service-based system design using UML. IEEE Trans. Softw. Eng. **36**(3), 371–389 (2010)
17. Wong, P.Y.H., Gibbons, J.: A process semantics for BPMN. In: Liu, S., Maibaum, T., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 355–374. Springer, Heidelberg (2008)
18. Ye, J., Sun, S., Song, W., Wen, L.: Formal semantics of BPMN process models using YAWL. In: Second International Symposium on Intelligent Information Technology Application, December 2008, vol. 2, pp. 70–74 (2008)