# Requirement Driven Service Composition: An Ontology-Based Approach

Guangjun Cai[1,2]

[1] The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[2] Graduate University of Chinese Academy of Sciences, Beijing 100049, China
caiguangj@mails.gucas.ac.cn

**Abstract.** Service-oriented computing is a new computing paradigm that utilizes services as fundamental elements for developing applications. Service composition plays a very important role in it. This paper focuses on service composition triggered by service requirement. Here, the processes modeling the requirement should be treated in parallel with describing service and a same ontology should be adopted for allowing the understanding between the requirement and services. An effect-based approach has been proposed based on our previous work on service description. This approach could be promising for tackling the challenge of services composition.

**Keywords:** Service-oriented computing, environment ontology, service composition, service discovery.

## 1 Introduction

Service-oriented computing (SOC) is a new computing paradigm that utilizes services as fundamental elements for developing applications [1]. Web service composition, which aims at solving complex problems by combining available basic services and ordering them to best suit the requirement and can be used to accelerate rapid application development, service reuse, and complex service consummation [2], is key to the success of SOC.

Many works, including standards, languages and methods, have been done to promote web service composition. But facing the challenge of high complexity composition problem with massive dynamic changeable services and on-demand request, most of them fail to provide an effective solution. Most approaches, such as [3] and [4], have only considered the requirement described by IO or IOPE. Second, few composite approaches use requirement as part of the composite service building process. Though the approach introduced by [4], [5]) consider the request in the composition process, the former fail to address the behavior and the latter leave all the task of requirement description to user. Consequently, they cannot cope with the challenge which the rapid change of user demands.

Different from them, we think that the composition process indeed relates to the requirement and user, the requirement should play a greater role in a service-oriented computing paradigm. Thus, it is necessary to consider the problem what the requirement

Z. Shi et al. (Eds.): IIP 2010, IFIP AICT 340, pp. 16–25, 2010.

is in detail. This paper proposes a requirement-driven services composition method. We use environment ontology as the unified description of web service and requirement. From this starting point, a process modeling the requirement is processed and the environment ontology is adopted for allowing the understanding between them. Based on these effect-based structured descriptions, we design the algorithm for decomposing the requirement in order to identify suitable component services and take it as the basic step for composing service. It not only helps reduce the complexity of the problem and to improve concurrency, but also provides flexibility among various part of the requirement. And then we present the discovery algorithm for the precise matching by utilizing the unified explicit formal semantic description. Finally, we end with a method generating the composition service.

For the space limitation, this paper concentrates on the behavior aspect of service or requirement in a single domain. The rest of this paper is structured as follows. In section 2, we introduce environment ontology and a usage scenario. Section 3 proposes the base work of composing a web service, the descriptions of a web service or a requirement. Section 4 presents the algorithms decomposing the requirement, discovering suitable service and generating composition service. After discussing some related work in section 5, we conclude in section 6.

## 2 Environment Ontology

Environment ontology, which is proposed according to the thinking of requirement engineering based on the environment modeling, is extended further in [6]. In this approach, the environment of a Web service is viewed to be composed of those environment entities that the Web service can interact with. The concepts and associations of environment ontology are shown in figure 1. These associations form a general conceptualization of the particular entity. Entity type a web service imposed on is suggestive of some property of the service.

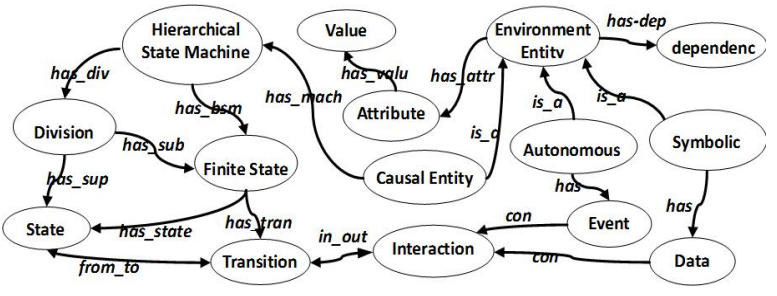


**Fig. 1.** Associations of environment ontology

In this paper, we embed a usage scenario for a travel system as an example. Part of the environment ontology about this domain is shown in figure 2, where the left figure show the environment entities and the relation among them, the right one is the part description about a causal entity ticket.

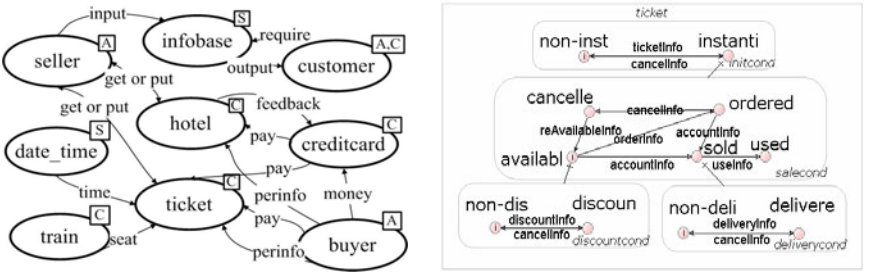


**Fig. 2.** Environment ontology of travel domain

## 3 Services Description and Requirement Description

Service description and requirement description is the prerequisite of service composition. Moreover, full automatic service composition needs a complete, formal specification. According to [3], it is difficult to provide a behavior-based requirement description for user.

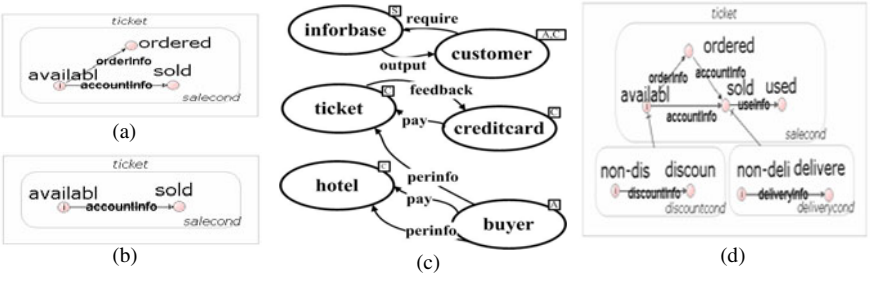


**Fig. 3.** Descriptions of service and requirement

Instead of focusing on the services of their own, environment ontology-based service description give attention on the effects imposed on their environment entities. The capabilities of services are expressed by the traces of the changes of the environment entities caused by the interactions of the services with these entities. We have presented the method for describing services[6]. Here some description results about ticket are presented in figure 3(a, b). They are part of environment ontology of the ticket and illustrate the changing process a service supports.

The process (figure 4) to describe the requirement is similar with that depicts web services, besides needing to facilitate the user to input and modify user information by using ontology. At the same time, the description result can be seen as the effect description of the composing web service if succeeded. The description result is also illustrated in figure 3, where part (c) shows the environment entities and the relations among them, and part (d) presents the requirement about the environment entity ticket.

```
ReqSpecification(userInfo, Onto, req) //requirement description based on environment ontology
Input: userInfo, Onto
Output: req
1       UserPro:=Generate(userInfo,Onto);  //generate user profile in interaction manner,
                //which including environment entities selection, output and target states selection,
                // input and initial state selection
2       req:=generate(Onto, UserPro);// similar with  the algorithm generating service description
3       if(ismulti(req)){
4          reqTem:=trim(req,userInfo);//determine the req according to user information
5          req:=ReqSpecification(UserInfo, reqTem, req);
6       }
7       if(req=∅){
8          userPro:=Modify(UserPro, Onto);
9          req:=ReqSpecification(UserInfo, Onto, req);
10      }
11      return req;
```

**Fig. 4.** Pseudocode of the requirement decomposition among environment entities

## 4 Services Composition

The task of web service composition is to search appropriate service and to arrange them in suitable order. Based on the environment ontology, we present a method through decomposing the requirement to search the available service and using the precise matching rules to judge. Thus, we divide the task into three parts: decomposing the requirement, discovering the suitable component services, generating the composition service.

### 4.1 Decomposing the Requirement

In our method, service requirement is described in a formal detailed way. It means that the same functional services have the same unique description. So we can acquire a composite service through a way which firstly decomposes the requirement into various parts to discover and compose the respecting services. The decomposing algorithms are introduced below.

**Decomposition based relations among environment entities.** The essence of the task in this section is to classify the set of environment entities, which need not consider the internal changing process in the environment entity under the effects of services. The behavior of decomposition among environment entities is summarized by the algorithm listed in figure 5, where req and reqSet denote the input and output of each algorithm, respectively. The requirement is firstly divided by DivByEntDep() according to the dependability among entities. The reason is that there is no dependence among different sub requirements. Secondly, we decompose a requirement by DivByEntIntTyp() according to the type num of dependences or entity type. The reason is that there is little constrain than other situation. Finally, the decomposition is

```
    DivByEntDep(req, reqSet) //req contains unrelated environment entities
1.    while(EnvEntNum(req) ≠ 0) {//req contains two or more environment entities
2.      e:=getEnvEnt(req);//gets environment entity from req
3.      if(Relate(e)≠EnvEnt(req)){// Relate(e) is a set of environment entities which directly or
            //indirectly depend on the environment entity e
4.        eReq:=create(Relate(e), req);//create() generate the requirement responding to relate(e)
5.        reqSet:= reqSet ∪ {eReq};
6.        req:= Remove(Relate(e), req);//removes the requirement corresponding to relate(e)
7.      }//end if
8.    }//end while

    DivByEntIntTyp(req, reqSet) //req contains many related entities
11.   if(∃e((e∈ Req) ∧ (inDep(e)=0 ∨ outDep(e)=0 ∨ type(e)=A ∨ type(e)=S))){
        //inDep(e), outDep(e)repsents  the number of dependents of environment entity e
        //type(e)  represents the type of e
9.        newreq:= create(e, Req);
10.       ReqSet:= {newreq} ∪ DivByEntDep (req-newreq, reqSet);
11.   }

    DivByEnt(req, reqSet) //each req in reqSet contains only one environment entity
12.   for(each e in Req){// e represents environment entity
13.       ReqSet := ReqSet ∪ create(e, Req); //create() generate the requirement responding to e
14.   }//end for
```

**Fig. 5.** Pseudocode of the requirement decomposition among environment entities

done by DivByEnt() based on environment entity. That is because there is only message dependence among different entities.

**Decomposition in the environment entity.** The requirement decompositions in the environment entity vary by type of entity. The decompositions in autonomous entity and symbolic entity, similar with transition-based decomposition in causal entity, can be directly done based on their basic behavior event or data by the algorithm DivByBeha()in figure6. But for causal entity, we need to consider its hierarchical structure in detail.

```
    DivInHsm(req, reqSet) //req contains exactly one hsm
1.    rootfsm:=GetRootfsm(Req);//rootfsm, a finite machine having no super state
2.    for(each subhsm of rootfsm){//subhsm.rootfsm.superstate∈ rootfsm
3.      ReqSet:=ReqSet ∪ create(subhsm, Req);// create(subhsm, Req) generate
            // the requirement responding to subhsm
4.    }
5.    ReqSet:=ReqSet ∪ create(rootfsm, Req); // create(rootfsm, Req) generate
                    // the requirement responding to rootfsm
```

**Fig. 6.** Pseudocode of the requirement decomposition in environment entity

```
    DivInFsm(req, kstate, reqSet) //req contains exact one fsm
7.    if((req.State-(From(kstate) ∪ To(kstate)))=∅){ //req.State represents all the state of req
8.      From(kstate):=From(kstate)-{kstate}; //From(kstate) represents all the states kstate can reach
9.      To(kstate):= To(kstate)-{kstate};//To(state) represents all the states which can reach kstate
10.   }//end if
11.   req1=create(req.State-(From(kstate) ∪ To(kstate)), req); //the req in different branch
12.   req2:=create(From(kstate)-To(kstate), req);//the req before the kstate or the loop containing it
13.   req3:=create(To(kstate)-From(kstate), req);//the req after the kstate or the loop containing it
14.   req4:=create(From(kstate)∩To(kstate),req);//the req in the same loop with kstate
15.   reqSet:= {req1} ∪ {req2} ∪ {req3} ∪ {req4};

    DivByBeha(req, reqSet) //each req in reqSet only contains one basic behaviour
16.   for(each b in req){//b can represent a transition, data or envent
17.       reqSet := reqSet∪ {create({b}, req)};
18.   }//end for
```

**Fig. 6.** *(continued)*

For a requirement in one causal entity, we firstly divide it using the algorithm DivInHsm(), based on the reason the requirement of the finite state machines at the same sub-tree has greater relevance than at different sub-trees. The decomposition process could be repeated until finding available service or reaching that each requirement only contains one finite state machine. Then, we decompose the requirement according to the key states, which can be divided into two classes. The first class includes that the initial, middle or target states in it, the second contains the states which is the super state of some finite state machine. The reason for this is that for user should have capabilities to choose what to do next step on the key state, the functionality on both side of the state could be separable. If there were still no available services for the result requirement, the algorithm DivByBeha() is used.

**Table 1.** Decomposition result of each step by the decomposition algorithm

| Algorithm | Decomposition result |
|---|---|
| DivByEntDep | {infobase, customer},{hotel, creditcard, ticket, buyer} |
| DivByEntEntTyp | {infobase},{customer},{buyer},{hotel},{creditcard, ticket} |
| DivByEnt | {infobase},{customer},{buyer},{hotel},{creditcard}, {ticket} |
| DivInHsm | {salecond}, {discountcond},{deliverycond} |
| DivInFsm | {available, ordered, sold},{sold, used} |
| DivByBeha | {available, ordered}, {ordered, sold}, {available, sold},{ sold, used} |

Taking the requirement in figure 3(d) as an example to present how to use these algorithms. For illustrating how to use them, we assume that each match fails below. The decomposition results are listed in table 1, where the set of entities, fsms and states denote corresponding requirement in different level requirement, respectively.

### 4.2 Selecting the Suitable Component Services

Service discovery, which enables a service requester to locate counterpart, plays a critical role in web service composition. With the increasing number of web services with similar functionality, measuring the match degree will become more and more important. However, there is a gap in most current approaches between service advertisement and service requirement. Environment ontology facilitates this, not only providing a unified semantic description, with additional knowledge about context, but also prompting to mark descriptions with the weight on modularity process ontology. Based on this description, we propose a precise match degree, as a sound criterion, to measure and select service. The functionalities can be measured by various aspects, such as the number of transitions, events, or data. The formulas calculating the matching degrees are shown as follows:

$$\text{ComDeg}=(\text{UseSerFun}/\text{ReqFun})\times 100\%. \tag{1}$$

$$\text{NecDeg}=|\text{UseSerFun}|/(|\text{UseSerFun}|+\text{NumUseFun})\times 100\% \tag{2}$$

$$\text{ValDeg}= (\text{UseSerFun}/\text{SerFun})\times 100\% \tag{3}$$

$$\text{GloDeg}=\text{Wc}\times\text{ComDeg}+\text{Wn}\times\text{NecDeg}+\text{Wv}\times\text{ValDeg} \tag{4}$$

In the formulas above, SerFun, UseSerFun and ReqFun denotes all the functionalities, the useful functionalities for the requirement a service can provide and all the functionalities the requirement needs. Moreover, NumUseFun represents the number of the functionalities in UseSerFun which other services can provide; Wc, Wn and Wv represent the weights of ComDeg, NecDeg and ValDeg respectively and the sum of them equals 1.

Based on the definition of the matching degree, we describe a service discovery method in figure 7. The service is firstly selected according to their globe matching degree in line 4. Then based on the matching type of a service, we process them in corresponding method. Taking the available services a, b in figure 3, the requirement

```
Discovery(req, serSet, t, service, subSerSet)
Input: req, serSet, t; //t represent the threshold between the req and the service
Output: service, subSerSet;//subSerSet is be used when service is nil
1   if(serSet=∅){return nil;}
2   for(service in serSet){
3      campares services with req and calculates their matching degree;
4      if(gloDeg>t){matchSerSet:= matchSerSet ∪ {service}; }
5   }
6   while(matchSerSet≠∅){
7      chooses matSer with necDeg=100% or with maximal gloDeg;
8      if(matSer.comDeg=100%∧ matSer.valDeg=100%)//exact matching
9        return service;
10     if(matSer.comDeg=100%∧service.valDeg<100%){//subsume matching
```

**Fig. 7.** Pseudocode of the component services discovery

```
11        if(isdivide(matSer, req)){ return divide(matSer, req);}
12        else {continue;}
13      }//end subsume matching
14     if(matSer.comDeg<100%){ //interaction or plug-in matching
15       add matSer in subSerSet prepare for future choosing;
16     }//endif
17  } //end while
```

**Fig. 7.** *(continued)*

d in figure 3 as an example, ComDeg, NecDeg and ValDeg of a is 2/6*100%, 2/(1+2)*100% and 2/2*100%, respectively. After getting the above values, we can easily calculate GloDeg of it to choose service.

### 4.3 Generating the Composition Service

The task of this step is to determine the relation among selected component service according to the relations between functionalities in a requirement and the relations between each component service with the responding sub requirements. The Pseudocode is listed in figure 8, where Req, Onto, serSet and t denote input, ComSerModel denotes the output.

```
GenComModel(Req, Onto, serSet, t, ComSerModel)
Input: Req, Onto, serSet, t
Output:ComSerModel
1   ReqSet.add (Req); //inserts the copy of req into ReqSet
2   while(ReqSet≠∅){
3     for(each req in ReqSet){
4       using discovery(req, serSet, t, subSerSet) discovery service and generate subSerSet;
5       ReqSet:=ReqSet-{req};
6       if(service≠nil){// discovery success
7         LabeledReq:=LabeledReq ∪ label(req, service.name); //labels Req using service.name
8         break; //end for
9       }
10      if(subSerSet=∅)//req cannot be satisfied by service in serSet
11        return nil; // composition failure
12      newreqSet:=divide(req, choosealg(ruleset));//divides req according the selected algorithm
13      ReqSet:=ReqSet ∪ newreqSet -{req};
14    }//end for
15  }//end while
16  ComSerModel:=Generate(LabeledReq, req);
```

**Fig. 8.** Pseudocode of the composition service generation

We have simulated our method in a Java platform, where the worst-case time complexity is shown in table 2. In it, |ent|, |beh|, |tran|, |event|, |data| and |ser| denote the number of the behaviours, environment entities, transitions, events, data and services.

Moreover, m denotes the repetitions, and n shows the times for discovering all the requirements, and Lev(hsm) denotes the average level number of all the hsms. Hence, the overall complexity of our method is at the polynomial level.

**Table 2.** The complexity degree of each algorithm

| Algorithm | Des | Div1 | Div2 | Div3 | Div4 | Div5 | Div6 | Div7 | Dis | Com |
|---|---|---|---|---|---|---|---|---|---|---|
| Worst-case complexity | $m\times \|beh\|^3$ | $\|ent\|^2$ | $\|ent\|^2$ | $\|ent\|^2$ | $\|ent\|$ | Lev (hsm) | $\|tran\|^3$ | Max(\|data\|, \|tran\|,\|event\|) | $\|ser\|\times \|beh\|$ | $n\times$ T(dis) |

## 5 Related Work

Web service composition is a research topic attracting attention daily. The differences between our method with others are illustrated in table 3, where I, O, P, E, B and "-" denote input, output, precondition, effect, behaviour and unspecified explicitly, respectively.

**Table 3.** Comparison of various approaches to service composition

| Approach | Service | Request | Content | Composition method |
|---|---|---|---|---|
| McIlraith [4] | IOPE | IOPE | - | agent |
| Hamadi [9] | IO | - | - | - |
| Bultan [10] | B | B | conversation | behavior equivalence |
| Fensel [11] | IOPEB | IOPEB | service, goal | mediator |
| Maamar [12] | IOB | IOB | context | agent technology |
| Berardi [5] | B | B | - | behavior equivalence |
| Sirin [7] | IOPE | IOPE | service | complex service-based decomposition |
| Brogi [3] | IOB | IO | - | Graph-constructing and coloring |
| This paper | IOPEB | IOPEB | environment | Requirement-driven |

## 6 Conclusions

This paper proposes that the essence of the composition of web services is the combination of effects of these services on their environment, and illustrates the requirement can play more important role in it. Compared with the existing efforts in this field, this work advances the state of art in the following aspects:

–The sharable environment ontology serves as a common knowledge background of both the services and the requirement. That enables the capability matching at semantic and behavior level.

–The ontology-based requirement description method reduces difficulty of requirements description as well as provides a more understandable and more expressive specification.

–The structured effect-based requirement specification prompts hierarchical effective decomposition and composition-oriented service discovery.

This paper describes an on-going work for tackling the issue of automatic service composition. In the next step, we will extend the ontology for supporting the service composition with different granularity and also in various domains. And then we will enhance the service composition procedure for considering the non-functional concerns. Moreover we will also focus on the verification of the capability profiles for the correctness of the composite services.

**Acknowledgments.** This work is partially supported by the National Natural Science Fund for Distinguished Young Scholars of China under Grant No.60625204, the Key Project of National Natural Science Foundation of China under Grant No. 60736015 and 90818026, the National 973 Fundamental Research and Development Program of China under Grant No. 2009CB320701.

## References

1. Papazoglou, M.P., Georgakopoulos, D.: Service-Oriented Computing. Communications of the ACM 46(10), 25–29 (2003)
2. Nikola, M., Miroslaw, M.: Current Solutions for Web Service Composition. IEEE Internet Computing 8(6), 51–59 (2004)
3. Brogi, A., Corfini, S., Popescu, R.: Semantics-Based Composition-Oriented Discovery of Web Services. ACM Transactions on Internet Technology 8(4), 19, 1–39 (2008)
4. McIlraith, S., Son, T.C.: Adapting Golog for Composition of Semantic Web Services. In: 8th International Conference on Knowledge Representation and Reasoning, Toulouse, France, pp. 482–496 (2002)
5. Berardi, D., Calvanese, D., Giuseppe, D.G., et al.: Automatic Composition of Transition-based Semantic Web Services with Messaging. In: 31st International Conference on Very Large Databases, VLDB Endowment, Norway, pp. 613–624 (2005)
6. Puwei, W., Zhi, J., Lin, L., Guangjun, C.: Building towards Capability Specifications of Web Services Based on an Environment Ontology. IEEE Transactions on Knowledge and Data Engineering 20(4), 547–561 (2008)
7. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN Planning for Web Service Composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2004)
8. Martin, D., Burstein, M., Hobbs, J., et al.: OWL-S: Semantic Markup for Web Services. The OWL Services Coalition (2004), http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/
9. Hamadi, R., Benatallah, B.: A Petri Net-based Model for Web Service Composition. In: 14th Australasian Database Conference, pp. 191–200. Australian Computer Society, Australia (2003)
10. Bultan, T., Fu, X., Hull, R., Jianwen, S.: Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In: 12th International World Wide Web Conference, Hungary, pp. 403–410 (2003)
11. Fensel, D., et al.: Ontology-based Choreography of WSMO Services. WSMO Final Draft (2007), http://www.wsmo.org/TR/d14/v0.4/
12. Maamar, Z., Mostefaoui, S.K., Yahyaoui, H.: Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. IEEE Transactions on Knowledge and Data Engineering 17(5), 686–697 (2005)