International Journal of u- and e- Service, Science and Technology
Vol.8, No. 6 (2015), pp.277-290
http://dx.doi.org/10.14257/ijunesst.2015.8.6.27

A Reliable Automatic Service Composition Method Based on Type Theory

Suichu Zhai[1], Zhong Xin[2] and Aihua Song

[1]*Hangzhou Power Supply Cooperation, Hangzhou, China*
[2]*Jilin Technology College of Electronic Information, Jilin City, China*
[3] *Hangzhou Dianzi University, Hangzhou, China*
* *Corresponding author: yinyuyu@hdu.edu.cn*

Abstract

Web Service composition has become a critical technology to react to the complex functional requirements put forward by business logic in e-commerce. How to guarantee the reliability of composite Web service is a hot topic through all the phase of lifecycle, including design, development and deployment. To this end, we are motivated to propose a Type Theory (TT) based method to automatic service composition. First of all, a service model in form of TT representation defines the atomic services declarations, semantic relation and composition request. Then, a set of TT rules that helps to prove the satisfiability of the user's request is formally given. Later, several useful tools transforming the service declaration files to the propositions and extracting the composition result from the proof is discussed. To face the challenge of reliability, we show a hybrid model, which dynamically binds service with the corresponding strict proving, to ensure the reliability of our method during compositing services. At last, the proposed method is implemented and a complete case study is conducted to demonstrate the applicability and effectiveness of our techniques.

Keywords: *web service composition, composition engine, type theory*

1. Introduction

Service-Oriented Computing (SOC) is a set of basic principles for designing and developing software in services industry. Each service is well-defined with business functionality which exposes its input and output information for potential invocations. Presently, there are many platforms and languages had proposed in literatures that allow to easily using heterogeneous Web Services online. The standards, such as Universal Description Discovery and Integration (UDDI), Web services Description Language (WSDL), Simple Object Access Protocol (SOAP) and Ontology Web Language for Services (OWL-S), have defined the ways to service description, selection and composition. However, service composition is used mainly to satisfy the complex requirement by composing some services together, including atomic service or composite services. The most promising method is called as Business Process Execution Language for Web Service (BPEL4WS) which focuses on representing composite service.

A large number of service composition methods have been proposed. They can be divided into three classes: workflow, AI-planning and theorem-proving. Manual operations are needed in workflow, especially in modeling step, namely workflow methods are not automatic. In the contract, the AI-planning methods and theorem-proving methods are automatic. For the former, the AI-planning method, however, performs badly when mass services involved because of the state-explosion. For the latter, the most theorem-proving methods perform better in reliability. Nevertheless, most AI-planning and theorem-proving methods can handle the semantic information of interfaces outside of the logic framework, which reduces reliability.

ISSN: 2005-4246 IJUNESST

In this paper, a web service composition method of theorem-proving is proposed, based on Type Theory (for short, TT). The main process of our method is transforming atomic services descriptions and semantic relation of interfaces into propositions in the form of axioms in TT. And the specification of user's requirement is formatted as conjecture expression. The way to compose web service request is to prove the conjecture by the previously mentioned propositions. In summary, this process of transforming and proving are automatic.

Our method faces the challenge of reliability in transforming the semantic relations and services declarations into propositions, then proving the conjecture requirement representation, automatically in TT. Both of the semantic relations and service connections have been formalized and proved in this process.

The three main contributions of our work are listed as follows.

(1) We propose a reliable automatic method using TT Prover to compose web service. Our method faces the challenges of reliability by proving the corresponding conjecture of the requirement strictly. Our research extends the research scope of TT.

(2) A hybrid service model is proposed in out method, which binds the concepts of web service, the logic proposition on type in TT. This model bridges proposition-proving and service composition. The model is different from other theorem-proving methods since any operation on it (including checking semantic similarity) must be formalized and proved, which guarantees the reliability of our method. What's more, this model can be used in not only service composition but also service selection and recommendation.

(3) We implement the method and build a complete system with lots of tools. The system can be used and extended in many research and application context.

The remainder is organized as follows: Section II summarizes the related works. Section III introduces the architecture of the system. Section IV introduces main technologies. Section V shows a case study and experiments. Section VII draws a conclusion and future research directions.

2. Related Work

Different researchers have different understandings of service composition [1]. Takes the view of workflow. Service composition is considered as connecting services together using some special rules to satisfy a business requirement [2]. Takes the view of program synthesis. Service composition is understood as integrating enterprise information systems (*e.g.,* ERP, OA, *etc.*,) and software. In [3], service composition is thought as finding a combination of some exciting services to satisfy the request of the user from the point of problem solving. However, in [4], service composition is thought as decomposing an entire task to some sub-tasks and finding each solution for these subtasks from the point of task planning.

The methods of existing models vary widely from one to another. Depending on the composition scheme generation way, web service composition methods can be divided into three categories: Workflow; AI-planning and Theorem-Proving.

Workflow service composition is based on traditional workflow technology. The difference between workflow and service composition is analyzed in [1, 5]. The likeness of workflow and service composition is that they both have a similar life cycle, which means they both have the modeling step and the running step. Data stream and control stream are needed to be designed in the modeling step. An engine analyses the definition of the data stream and control stream to build a runnable-instance in running step. However, there is still a huge gap between workflow and service composition. The atomic applications defined in the workflow are static, which means they will not be modified or removed frequently. The atomic services, however, are dynamic [6, 7]. There are many

famous techniques including eFlow [8], 9] and PPM [10]. These methods are not automatic, since the help of user is needed in modeling step.

AI-planning methods take the view of problem solving. The service composition problem is considered as an automated solving of the planning problem. Once a special initial state and a target state are clearly defined, the composition process is finding a suitable path from the initial state to the target state. PDDL [11] and Golog [12-14] are famous AI-planning method. Mayer in [15] presents a consistency based service composition approach, where service composition problems are modeled as the Constraint Satisfaction Problem (CSP). A composition framework named CWSF (Composition Web Service Framework), which is also based on CSP, is proposed in [16]. The composition methods based on heuristic search are mentioned in [17, 18]. A model, called Service dependency graph (SDG) allows modeling of complex and application-specific interfaces, which is presented in [19]. An extensibility model is proposed 3 years later, in [20], which specifies two kinds of dependence relations, one is operation dependence, and another is attribution dependence. In [21] - [26], many approaches models services using Petri nets. What's more, the semantic service composition is taken into account in [27-32]. AI-planning works automatically; nevertheless, the state-explosion is the bottleneck of these methods.

Theorem-proving, however, goes another way. Even though efficiency is the most difficult problem of all theorem-proving methods, many researchers are still doing more jobs in this field because of its relibility. One of the improvements is the method based on Linear Logic (LL), proposed by Rao [33-35]. It treats the service composition problem as the program synthesis problem. However Liner logic is not powerful enough in expression. Waldinger [36] proposed an idea for service synthesis by theorem proving. The main techniques used are automated deduction and program synthesis. Lämmermann [37] applies Structural Synthesis of Program (SSP) for automated service composition. SSP is a deductive approach to synthesis of programs from specifications. Yin [38] proposed a method using Martin-löf type theory to verify consistency and compatibility of web service inner behavior. In [39], he further proposed a new method to deal with Large-granularity Web Services Composition. Our research is based on this prior work. Comparing to other methods, our method formulas the semantic relations and includes them into our proving, which makes the semantic relation reliable.

3. Architecture

The architecture of our method is shown as Figure 2. The basic compositions of this system are as follows:

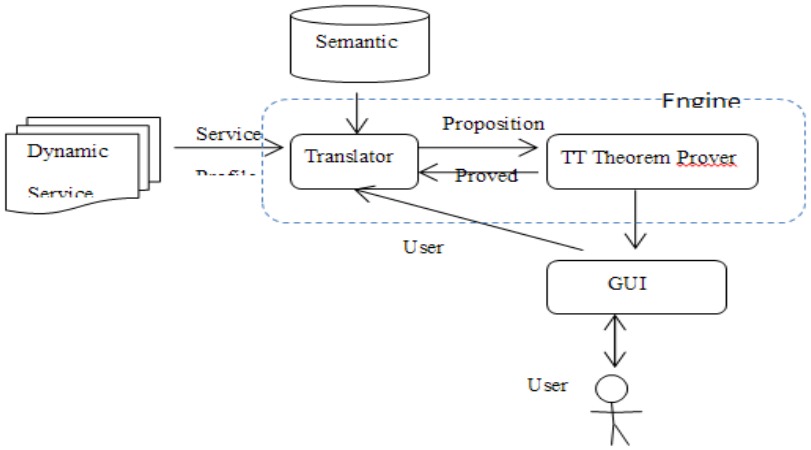

Figure 1. Architecture

Dynamic Service Library stores web services declarations. The Dynamic Service Library is used to provide necessary web services to satisfy the query of the user. The services are represented in the form of OWL-S (Ontology Web Language for Services). This library, however, is not static. It is easy to add and delete web services dynamically.

Translator is a critical component. Mainly, it has three functions. First, it transfers services profiles, which are in the form of OWL-S, provided by Dynamic Service Library into propositions in the form of expressions in TT. Second, it transfers the semantic relations of the interfaces into propositions. Third, it transfers the proved conjecture into BPEL4WS after proving.

Semantic Database is used to provide semantic ontology to analyze service profiles. The port of a web service can be defined as a concept, which is a node of the ontology tree. The subtype relation and belonging relation are easy to be found by searching the semantic ontology tree. Subtype relation and belonging relation are both used as propositions to prove targeted conjecture

TT Theorem Prover is the core component of this system. Essentially, it works as a theorem prover. Different prove tactics (*i.e.,* Distribution rate of conjunction) and existing proposition (*i.e.,* Atom Web Services) will be used to prove the conjecture. If the proof exists, the composition plan is contained in this proof. The proof will be transferred into a BPEL4WS file by the translator. If the proof doesn't exist, it means the composition request cannot be satisfied in current service context. Coq is used as the proving-tool in the prover.

GUI helps user typing in the query specification into the system. The composition result is also shown by the GUI.

A simple composition process is as follows. At first, user types in request specification with the help of GUI. Then the specification is transferred to a conjecture. The Dynamic Service library provides corresponding atomic services in the form of OWL-S. The translator transforms these atomic services to propositions. Subtype and Belonging relations are also transformed into propositions with the help of Semantic Database. TT prover tries to prove the conjecture using the propositions. If success, the result will be converted into BPEL4WS by Translator. Otherwise, the failure message will be shown by the GUI.

4. Primary Technology

4.1. Type Theory

Type Theory is a logical system and a set theory based on the principles of mathematical constructivism. The Intuitionistic type theory was introduced by Per Martin-Löf, a Swedish mathematician and philosopher, in 1972.

The basic idea of TT is the Curry-Howard isomorphism, which has two meanings. First it combines proposition and type together. Second, it combines proof and program together. For example, the expression $\delta:\Delta$ can be understood in four ways. First, δ is an element belongs to set Δ. Second, δ is a proof of proposition Δ. Third, δ is a program satisfied specification Δ. Forth, δ is a solution of problem Δ. Thus, it is possible to describe the composition problem in many different levels with TT. This feature, on the other hand, makes TT a powerful tool in formalizing. The combination of the concepts of type, proposition and service brings two important advantages as follows. Because of these advantages, TT is chosen as the logic foundation of our method.

(1) The process of reducing semantic relation and composing service can both be formalized in TT. They both can be representing as the theorem-proving.

(2) Obviously, we can compose web service by proof the corresponding conjecture.

In the formalization of TT, like other formal method theory, more complex types are constructed by limited basic types (such as Nat, means natural number type) and type

connectives (such as Π, Σ). The corresponding of type connectives and normal logic connectives are shown in Table 1:

Table 1. Logic Connectives and Corresponding Type Connectives

Logic	Type
A /\ B	Product type
A \/ B	Sum type
A -> B	Function type
	Dependent Product type
	Dependent Sum type
False	Empty type
True	Unit type

The syntax of the TT fragment is presented by the following grammar

The fragment T consists of basic type t, product type, sum type, arrow type, unit type and empty type. The arrow type is quite similar to the function (function type) in Meta Language (ML). In fact the arrow type is a special product dependent type. Since type and proposition are two sides of a coin of the formal system of TT. Analogously we define the logic proposition by the following formula:

P consists of basic proposition p, conjunction disjunction, arrow, False and True. Some inference rules are defined as follows:

Table 2. The Inference Rules of TT

Name	Rule
Prop rule	
Constant rule	
Type rule	
Variable rule	
rule	
rule	
rule	
App rule	
Pair rule	
rule (subtype rule)	

4.2. Web Service Model

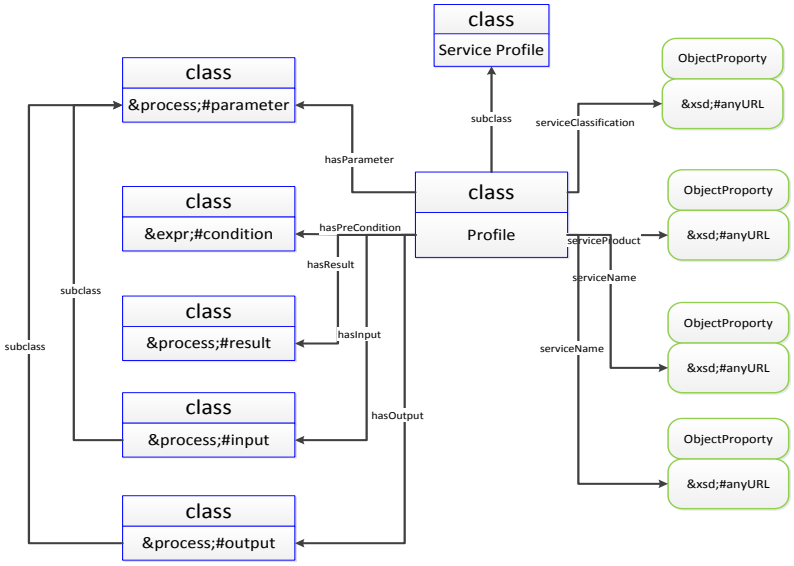

Figure 2. The Upper Module of Web Service

Figure 2 is the upper ontology of web service. A Web Service (WS) has many properties including service name, cost, description, information about provider and some parameters including condition, input, output, etc. There are mounts of standards of WS, such as WSDL (Web Service Description Language), DAML-S (DARPA agent markup language for services) and OWL-S (Web Ontology Language for Services). Generally, a web service can be expressed by the following TT formula.

In this formula, is the set of propositions representing Web Services, namely the context. Represents the conjunction of non-functional constraints of service. Represents the conjunction of non-functional results of the service. is the functional part of a service. In this formula, the symbol I represents a conjunction of input parameters, O represents a conjunction of output parameters and E represents a conjunction of exception. This formula can be explained as follows: in the context of and non-functional constraints, there is a process and non-functional result.

```
<process:AtomicProcess rdf:ID="Query">
    <process:hasInput>
        <process:Input red="&onto,#PName">
            <process:parameterType rdf:about="&xsd,#string">
        </process:Input>
    </process:haInput>
    <process:hasOutput>
         <process:Output red="&onto,#GID">
             <process:parameterType rdf:resource="&xsd,#number">
         </process:output>
    </process:hasOutput>
    <process:hasOutput>
        <process:Output red="&onto,#PPrice">
             <process:parameterType rdf:resource="&xsd,#number">
         </process:output>
  </process:hasOutput>
    <process:hasOutput>
        <process:Output red="&onto,#PException">
             <process:parameterType rdf:resource="&xsd,#exception">
         </process:output>
     </process:hasOutput>
</process:AtomicProcess>
```

Figure 3. Part of OWL-S File

Figure 3 presents a sample of an OWL-S file, which declares a standard service named Query. An operation is defined in Query. An input message named &onto, #PName, two output messages named & onto, #GID, & onto, #PPrice and an exception message named &onto, #PException are declared in this OWL-S file. The symbol I mentioned in the TT formula represents the input message & onto, #PName, while O represents the output messages: & onto, #GID+&onto, #PPrice and E represents the fault message &onto,#PException.

Even though there are some non-functional properties. It makes sense just paying attention to the functional part, because the non-functional properties can be handled in a similar way. In the following, we will discuss the functional part mainly.

4.3. Transforming Services Into Propositions

In our system, Web Services are represented by OWL-S. The following OWL-S example shows the way of transforming.

Figure 3 presents part of a sample OWL-S file. This service has an input named PName whose type is &xsd;#string, and two outputs named GID and PPrice which are both entities of type &xsd;#number.

The concept of type here is different from what we discuss in TT. It is better to be treated as a basic type. In most theories, the concept of type is suitable when distinguishing type “string” with type “number”. However, this is not enough in TT. As we can see, the type “string” has only restricted this parameter in syntax, rather than semantics. It is impossible to distinguish the difference by this simple concept of type in the syntax. It is too weak in complex web service system. In our method, we propose a “stronger” type, which is exactly the concept of type in TT. We use the parameter name as its type name, which is certainly unique in the system. We can transfer the OWL-S profile mentioned in Figure 4 to the formula:

The type Query is defined in ontology &onto. So we can get a shorter formula below with &onto removed.

This formula is exactly a proposition in TT. There are many ways to understand this formula. First, finding a service which has an input named PName and two outputs named GID and Price. Second, looking for a function which has an input named PName and two outputs named GID and Price. Third, finding an item which belongs to type named. Forth, finding a proof when proposition named PName exists, while the goal is the conjecture whose type is.

4.4. Transforming Semantic Relation Into Conjecture

The semantic relation we took into account is the subtype relation. An output of a service is suitable to be used as the input of another service when the types of these two interfaces are equal or have the relation of the subtype. For any $\delta: \Delta$, the sufficient and necessary condition of $\delta: \Theta$ is . This can be proved using subtype rule (defined in Table 2). For example, the type husky is a type, and the type dog is another type. The relation of husky and dog is the subtype relation (husky is the subtype of dog), which can be found in the ontology of animal. The subtype relation can be transformed as follows:

4.5. Transforming Request Into Conjecture

User types in the specification with help GUI. The requirement contains two basic parameters: list of inputs and list of outputs. An example of request of purchasing contains these parameters as follows:

Table 3. Parameters of Request

Parameter	Parameter type	Data type
&onto;#Bank_name	Input parameter	&Basic;#string
&onto;#User_account	Input parameter	&Basic;#string
&onto;#User_password	Input parameter	&Basic;#string
&onto;#Transaction_money	Input parameter	&Basic;#double
&onto;#Transaction_result	Output parameter	&Basic;#bool

The request contains four input parameters and one output parameter. The input parameter corresponds the service input. The output parameter corresponds the service output. The entire conjecture can represent as follows (it is quite similar to the model of service).

We can get a shorter formula below with &onto removed.

A full example will be shown in section V.

4.6. Proof as Composition

The main idea of our method is "proof as composition". To understand this idea clearly, let's consider what happened in the process of composing. From the view of construction, service composing is to construct a complex service, which satisfies the requirement of users, using the atomic services. The proving, on the other hand, is to construct a proof satisfying the proposition. According to the definition of type in TT, the specification of requirement can be represented as a complex type. The proof is an available instance of the type of requirement, which can be constructed from the existing instance of propositions. Therefore, the proof is exactly the construction result and the proving process is the constructing process. So it is natural to combine the service composition and proposition proving together. The problem of composition is now transformed into finding a proof of the corresponding proposition.

A proposition can be proved in TT prover in two ways: manual way and automatic way. The manual way is using the basic rules of TT on the existing propositions manually to construct the goal. In the contract, automatic way makes it possible to prove the exact target automatically. The automatic way is used in our method. An example of proving will be shown in section V.

This paper integrates the above HGLG algorithm into GridSim simulator, and adds the corresponding energy parameters to nodes to assess the performance of the algorithm. To verify the effectiveness of the algorithm, this paper will compare the performance and energy consumption among the non-integrating data partition algorithms cloud storage system (NPS), the integrating traditional classification algorithm cloud storage system (TDCS), and cloud storage system integrated high availability green gear-shifting mechanism (HGLG). This experiment assumes that the number of backup for NPS systems is 3, the number of hot data backup for TDCS system is 3, the number of cold data backup is 1, and also the new data and the seasonal hot data for TDCS system are all stored in a hot zone. The chapter compares different impact on performance and energy among three systems through different synthetic load, different proportion of new data and seasonal hot data, and different system scale to evaluate the strategy we proposed.

5. Case Study

In order to understand our method directly, let's consider an example of Groupon. Groupon is a new way of consumption, which groups people together to obtain the optimal price. Businesses can provide a lower price for Groupon rather than the retail.

Groupon is quite popular in recent year. People can participate in a Groupon in two ways. First, the one, called the sponsor, can register a Groupon himself/herself, and write down the target goods, expected price and least number of participators. The number is the least number. If the goods provider agrees, the Groupon information will present on the Internet. Those also wanting these goods could register to sponsor. Once enough number of people participates in this Groupon, the Groupon can make a deal, for which participators can thus get the intended goods in confirming price. Second, people can participate in existing Groupon. They apply for this Groupon, pay the confirmed price and wait for the number of participators reaches the threshold. At last the good will be sent to them in confirming price.

The ontology graph of Groupon is quite complex. Figure 5 shows a part of it. Let's consider a boy who wants to buy a book named <Thinking in Java> by Groupon. But there is not an existing service can satisfy his request. It is necessary to compose atom services together.

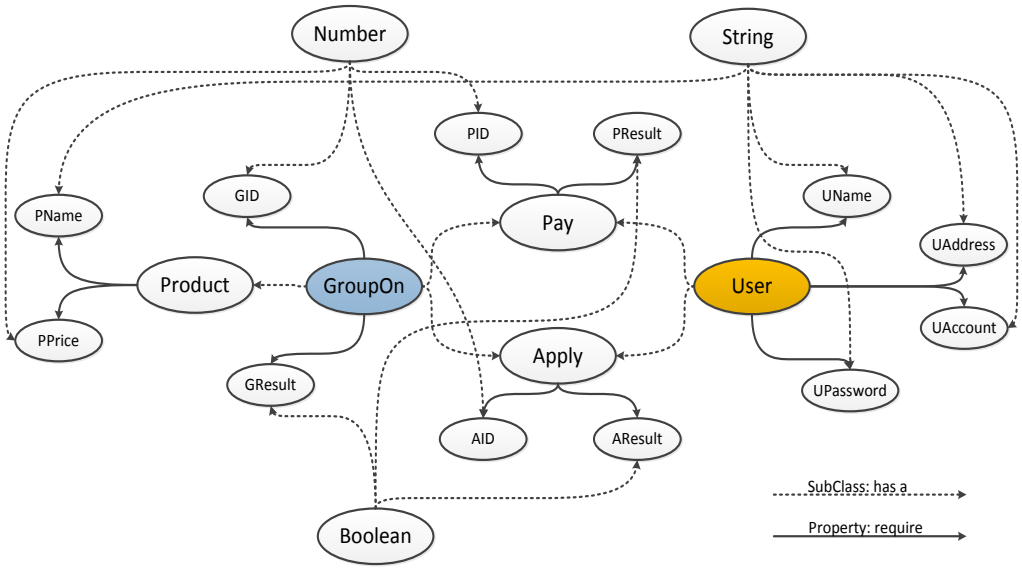

Figure 4. Part of the Ontology of Groupon

5.1. Atomic Services

Here we list some atomic services on the Internet.

Query—given product name, this service can query Groupon database to find out existing Groupon which purchases this product. It outputs the id of Groupon.

Apply—given Groupon id, the name and address of the user, the service can help to applying to this Groupon. It outputs the id of pay and the id of the application

Pay—given pay id, account and password of a user, the service is used to pay the application with the user's money in the bank on the Internet.

Check—given the result of pay and application id, the service will check whether it is successfully applied. It outputs the result of application.

Trade—given application id, this service will send the intended books to the user. It outputs the Groupon result.

5.2. Transforming Services

These atomic services are declared in our Dynamic Service Library. What we do first is the transformation. The converted propositions are as follows:

5.3. Transforming Request

The request as we described is quite complex. The boy wants to participate in a Groupon with an acceptable price. The input contains the name of the book, namely <Thinking in Java>, the boy's basic information, including name, address and bank account information, which is used to pay for the book. The target service, namely the request, can be present as a conjecture as follows.

5.4. Proving and Extraction

The proof in mathematical is shown in Figure 6. There are many inference rules used in the proof which were defined in Table 2. In our research, Coq is used as the kernel of TT prover. Coq is a theorem proving tool, which can be downloaded from http://coq.inria.fr/. The prove process is automatically completed in TT prover. The proof itself is represented by a functional language, which is shown in Figure 7. The last job is to transfer this proof to a BPEL4WS file. The result is shown in Figure 8.

```
/* Parameter Defintion */
1.  Parameter PName PPrice PResult GID UName UAddress PID AID UAccount UPassword AResult GResult:Prop.
2.  Section Sample.
/*Hypothesis list*/
3.  Hypothesis QueryService:PName->GID/\PPrice.
4.  Hypothesis ApplyService:GID->UName->UAddress->PID/\AID.
5.  Hypothesis PayService:PID->PPrice->UAccount->UPassword->PResult.
6.  Hypothesis CheckService:AID->PResult->AResult.
7.  Hypothesis TradeService:GID->AResult->GResult.
/* Goal Defintion */
8.  Lemma Goal_manual:PName->UName->UAddress->UAccount->UPassword->GResult.
9.  Proof.
/*Main part of proof*/
10. intros pname uname uaddress uaccount upassword.
11. apply TradeService.
12. apply QueryService in pname.
13. destruct pname as [gid pprice].
14. apply gid.
15. apply CheckService.
16. apply ApplyService.
17. apply QueryService in pname.
18. destruct pname as [gid pprice].
19. apply gid.
20. apply uname.
21. apply uaddress.
22. apply PayService.
23. apply ApplyService.
24. apply QueryService in pname.
25. destruct pname as [gid pprice].
26. apply gid.
27. apply uname.
28. apply uaddress.
29. apply QueryService in pname.
30. destruct pname as [gid pprice].
31. apply pprice.
32. apply uaccount.
33. apply upassword.
/* Proof end */
34. Qed.
```

Figure 5. Proving Process In Coq

Figure 5 represents the basic steps of proving the conjecture in Coq. The process basically consists of applying hypothesizes and destructing the conjunction of outputs. Since Coq is an interactive proving tool, the proving process is quite hard to read without running environment. What's more, a tactic named "tauto" helps to prove this proposition

automatically, which makes it possible to apply our composition method in a real environment.

$$\frac{\dfrac{\vdash Query:PName\to GID\times PPrice}{PName\vdash GID\times PPrice}(intro\ rule)\quad \dfrac{\vdash Apply:GID\to UName\to UAddress\to PID\times AID}{GID::UName::UAddress\vdash PID\times AID}(intro\ rule)}{PName::UName::UAddress\vdash GID\times PPrice\times PID\times AID}(app\ rule)$$

$$\frac{\dfrac{\vdash Pay:PID\to PPrice\to UAccount\to UPassword\to PResult}{PID::PPrice::UAccount::UPassword\vdash PResult}(intro\ rule)\quad \dfrac{\vdash Check:AID\to PResult\to AResult}{AID::PResult\vdash AResult}(intro\ rule)}{PID::AID::PPrice::UAccount::UPassword\vdash PResult\times AResult}(app\ rule)$$

$$\frac{PName::UName::UAddress\vdash GID\times PPrice\times PID\times AID}{PName::UName::UAddress\vdash GID\times PID\times AID}(elim\ rule)\quad PID::AID::PPrice::UAccount::UPassword\vdash PResult\times AResult}{PName::UName::UAddress::UAccount::UPassword\vdash GID\times PResult\times AResul}(app\ rule)$$

$$\frac{\dfrac{PName::UName::UAddress::UAccount::UPassword\vdash GID\times PResult\times AResul\quad \dfrac{\vdash Trade:GID\to AResult\to GResult}{GID::AResult\vdash GResult}(intro\ rule)}{PName::UName::UAddress::UAccount::UPassword\vdash GResult}(app\ rule)}{\vdash PName\to UName\to UAddress\to UAccount\to UPassword\to GResult}(gene\ rule)$$

Figure 6. The Sample Proof

```
Goal_auto =
fun (H : PName) (H0 : UName) (H1 : UAddress) (H2 : UAccount) (H3 : UPassword) =>
let H4 := QueryService H in
and_ind
  (fun (H5 : GID) (H6 : PPrice) =>
    let H7 := ApplyService H5 in
    let H8 := H7 H0 in
    let H9 := H8 H1 in
    and_ind
      (fun (H10 : PID) (H11 : AID) =>
        let H12 := TradeService H5 in
        let H13 := PayService H10 in
        let H14 := H13 H6 in
        let H15 := H14 H2 in
        let H16 := H15 H3 in
        let H17 := CheckService H11 in
        let H18 := H17 H16 in let H19 := H12 H18 in H19) H9) H4
      : PName -> UName -> UAddress -> UAccount -> UPassword -> GResult
```

Figure 7. The Proof Present in Coq

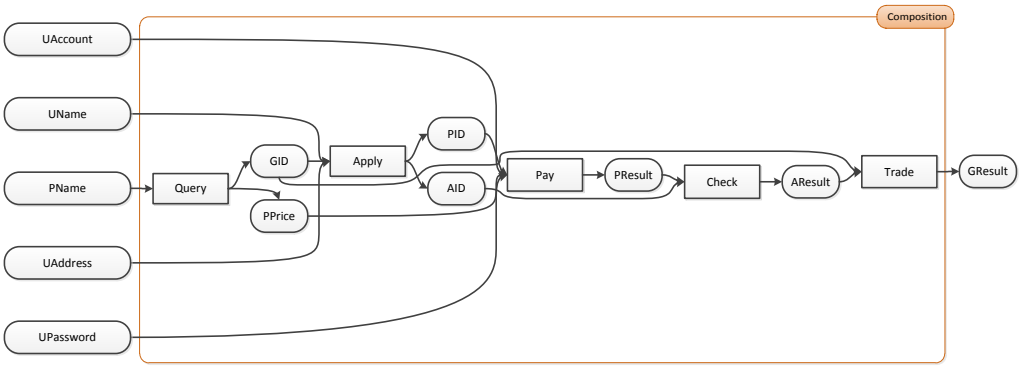

Figure 8. Composition Result

6. Conclusions and Future Work

Web service composition can satisfy the complex requirement of user. The method proposed in this paper faces the challenge of reliability using a hybrid model to bind service composition based on proposition-proving method. The reliability of composing process is ensured by the equivalence of proving and the formalization of semantic

relations. Then, complete composition architecture is proposed. Latter, atomic services are transformed into propositions. The request of the user is transformed into a conjecture. Finally, it is automatic to prove whether the existing services can satisfy the original request. Once the conjecture is proved as true, a composition process extracted from the proof can be used as trustworthy. The composition process will be transformed into WS4BPEL which is convenience to be used in today service application.

The future work will fall into two directions. One is that, a better model which can deal with a loop and switch control of the Web Service composition is needed to be studied. The second one is, more non-functional properties will be taken into account in our model.

Acknowledgment

The Project Supported by Zhejiang Provincial Natural Science Foundation of China under grant No.LY12F02029 and National Technology Support Program under grant No.2011BAH16B04.

References

[1] M. Paolucci, *et al.*, "Semantic matching of web services capabilities", The Semantic Web—ISWC 2002. Springer Berlin Heidelberg, **(2002)**, pp. 333-347.
[2] N. Srinivasan, M. Paolucci and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput", Proceedings of Semantic Web Service and Web Process Composition, **(2004)**.
[3] C. Zhou, L.-T. Chia and B.-S. Lee, "Service discovery and measurement based on DAML-QoS ontology", Special interest tracks and posters of the 14th international conference on World Wide Web, ACM, **(2005)**.
[4] S. Majithia, "Reputation-based semantic service discovery", Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE, 13th IEEE International Workshops on IEEE, **(2004)**.
[5] A. Bucchiarone and S. Gnesi, "A survey on services composition languages and models", International Workshop on Web Services–Modeling and Testing WS-MaTe, **(2006)**.
[6] Y. Syu, *et al.*, "Towards a genetic algorithm approach to automating workflow composition for web services with transactional and qos-awareness", Services (SERVICES), IEEE World Congress on IEEE, **(2011)**.
[7] M. B. Blake and D. J. Cummings, "Workflow composition of service level agreements", Services Computing, SCC, IEEE International Conference on IEEE, **(2007)**.
[8] F. Casati, *et al.*, "Adaptive and dynamic service composition in eFlow", Advanced Information Systems Engineering. Springer Berlin/Heidelberg, **(2000)**.
[9] F. Casati, *et al.*, "eFlow: a platform for developing and managing composite e-services", Research Challenges, Proceedings. Academia/Industry Working Conference on IEEE, **(2000)**.
[10] H. Schuster, *et al.*, "Modeling and composing service-based and reference process-based multi-enterprise processes", Advanced Information Systems Engineering, Springer Berlin/Heidelberg, **(2000)**.
[11] D. McDermott, "Estimated-regression planning for interactions with web services", Proc. AIPS, vol. 2, **(2002)**.
[12] S. McIlraith and T. C. Son, "Adapting golog for composition of semantic web services", PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING-INTERNATIONAL CONFERENCE-. Morgan Kaufmann Publishers 1998, **(2002)**.
[13] S. A. McIlraith, T. C. Son and H. Zeng, "Semantic web services", Intelligent Systems, IEEE, vol. 16.2, **(2001)**.
[14] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services", Proceedings of the 11th international conference on World Wide Web, ACM, **(2002)**.
[15] W. Mayer, R. Thiagarajan and M. Stumptner, "Service composition as generative constraint satisfaction", Web Services, ICWS, IEEE International Conference on IEEE, **(2009)**.
[16] E. Karakoc and P. Senkul, "Composing semantic Web services under constraints", Expert Systems with Applications, vol. 36, no. 8, **(2009)**.
[17] H. Meyer and M. Weske, "Automated service composition using heuristic search", Springer Berlin Heidelberg, **(2006)**.
[18] J. Xu and S. Reiff-Marganiec, "Towards heuristic web services composition using immune algorithm", Web Services, ICWS'08, IEEE International Conference on IEEE, **(2008)**.
[19] Y. Syu, *et al.*, "An automated workflow composition to semantic web services", Machine Learning and Cybernetics (ICMLC), 2011 International Conference on IEEE, vol. 2, **(2011)**.
[20] Z. Gu, J. Li and B. Xu, "Automatic service composition based on enhanced service dependency

graph", Web Services, ICWS'08, IEEE International Conference on IEEE, **(2008)**.
[21] D. Zhovtobryukh, "A Petri Net-based Approach for Automated Goal-Driven Web Service Composition", Simulation, vol. 83, no. 1, **(2007)**, pp. 33-63.
[22] R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition", Proceedings of the 14th Australasian database conference, Australian Computer Society, Inc., vol. 17, **(2003)**.
[23] Y. Tang, *et al.*, "SRN: An extended Petri-net-based workflow model for Web service composition", Web Services, Proceedings International Conference on IEEE, **(2004)**.
[24] W. Tan, Y. Fan and M. C. Zhou, "A petri net-based method for compatibility analysis and composition of web services in business process execution language", Automation Science and Engineering, IEEE Transactions on, vol. 6, no. 1, **(2009)**, pp. 94-106.
[25] Z.-Z. Qian, S.-L. Lu and L. Xie, "Automatic composition of Petri net based Web services", CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION, vol. 29, no. 7, **(2006)**, pp. 1057.
[26] Z.-Z. Qian, S.-L. Lu and L. Xie, "Automatic Composition of Petri Net Based Web Services", Chinese Journal of Computers, vol. 7, **(2006)**, pp. 007.
[27] S. McIlraith and T. C. Son, "Adapting golog for composition of semantic web services", PRINCIPLES OF KNOWLEDGE REPRESENTATION AND REASONING-INTERNATIONAL CONFERENCE-. Morgan Kaufmann Publishers 1998, **(2002)**.
[28] E. Sirin, J. Hendler and B. Parsia, "Semi-automatic composition of web services using semantic descriptions", 1st Workshop on Web Services: Modeling, Architecture and Infrastructure, **(2003)**.
[29] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes", The semantic Web-ICWS, **(2004)**.
[30] R. Zhang, R. I. B. Arpinar and B. Aleman-Meza, "Automatic composition of semantic web services", 1st International Conference on Web Services, **(2003)**.
[31] E. Sirin, B. Parsia and J. Hendler, "Template-based composition of semantic web services", Aaai fall symposium on agents and the semantic web, **(2005)**.
[32] K. Fujii and T. Suda, "Semantics-based Context-aware Dynamic Service Composition", ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 4, no. 2, **(2009)**, pp. 12.
[33] J. Rao and P. Küngas, "Application of linear logic to web service composition", In The First International Conference on Web Services, Las Vegas, **(2003)**.
[34] J. Rao, P. Kungas and M. Matskin, "Logic-based Web services composition: from service description to process model", Web Services, Proceedings. IEEE International Conference on IEEE, **(2004)**.
[35] J. Rao, "Semantic web service composition via logic-based program synthesis", Diss. Norwegian University of Science and Technology, **(2004)**.
[36] R. Waldinger, "Web agents cooperating deductively", Formal Approaches to Agent-Based Systems, **(2001)**, pp. 250-262.
[37] S. Lämmermann, "Runtime Service Composition via Logic-Based Program Synthesis", PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, **(2002)** June.
[38] Y.-Y. Yin, *et al.*, "Determining on Consistency and Compatibility of Web Services Behavior", Acta Electronica Sinica, vol. 3, **(2009)**, pp. 002.
[39] Y.-Y. Yin, *et al.*, "Proof as Composition: An approach for the Large-granularity Web Services Composition", JDCTA: International Journal of Digital Content Technology and its Applications, **(2010)**.

Authors

Suichu Zhai, received the M.S. degree from School of information engineering Northeast Dianli University, her major is computer application technology. He research interests focus on service computing and cloud computing.

Zhong Xin, He received the Master degree in Computer Application Technology from Northeast Dianli University in 2010. He is a currently a lecture in Jilin Technology College of Electronic Information, China. His research interests focus on senor network.

Aihua Song, she received the M.S. degree from School of Computer on Hangzhou Dianzi University, her major is computer application technology. She studies in the lab of Cloud Computing Technology Research Center, her research interests are service computing and cloud computing, mainly the energy-aware service composition.

