Validating trustworthy service composition through VIPLE and pi-calculus

Shenghui Zhao, Yuemin Li and Yang Wang

College of Computer Information and Engineering,
Chuzhou University,
Chuzhou, Anhui, China
Email: shzhao@ah.edu.cn
Email: lym@chzu.edu.cn
Email: wangyang@chzu.edu.cn

Yinong Chen*

School of Computing, Informatics and Decision System Engineering,
Arizona State University,
Tempe, AZ 85287, USA
Email: yinong@asu.edu
*Corresponding author

Abstract: The current formal verification practice focuses on functionality and does not consider verification of the non-functional attributes. In this study, we propose a method in which non-functional attributes are incorporated into the logic rules of inference in terms of composition of the linear logic and pi-calculus. Giving the credibility to non-functional attributes is important, especially in the cloud computing platform and IoT environments, where trust and security are ultra-important. Such studies have not been paid much attention by researchers and practitioners. In our approach, the evolvement of the non-functional attributes is included in the process of formal verification of the service composition scheme. In addition to theoretical analysis, we applied a tool named Visual IoT/robotics Programming Language Environment (VIPLE) to execute and verify the validity of the service composition model's function. We translate the proving process of the linear logic into the corresponding pi-calculus expressions. VIPLE can translate visual work flow into pi-calculus and can verify the correctness of pi-calculus expressions.

Keywords: formal verification; pi-calculus; service composition; visual programming; VIPLE.

Reference to this paper should be made as follows: Zhao, S., Li, Y., Wang, Y. and Chen, Y. (2020) 'Validating trustworthy service composition through VIPLE and pi-calculus', *Int. J. Simulation and Process Modelling*, Vol. 15, Nos. 1/2, pp.76–88.

Biographical notes: Shenghui Zhao received her PhD from Southeast University of China in 2013. She is a Professor at the School of Computer and Information Engineering and the head of the research group of cloud computing and big data at Chuzhou University in China. Her current research interests include web service, cloud computing, artificial intelligent and internet of things.

Yuemin Li received his Master's degree from Hefei University of Technology in 2009. He is a Senior Experimentalist at Chuzhou University in China. His current research interests include data mining, artificial intelligent and smart campus. He also conducts research on wisdom classroom.

Yang Wang received his Master's degree from the Jiangsu University of Science and Technology, in 2012. He is a Lecturer in School of Computer and Information Engineering at the University of the Chuzhou since 2013. He is involved in the foundation of the robotics laboratory in the school. He supervised student teams to win more than 30 awards in various robot competitions.

Yinong Chen received his PhD from the University of Karlsruhe (KIT), Germany, in 1993. He did a post-doctoral research at the University of Karlsruhe and LAAS-CNRS in France. From 1994 to 2000, he was a Lecturer and a Senior Lecturer in School of Computer Science at the University of the Witwatersrand, Johannesburg. He joined Arizona State University in 2001 and is a Principal Lecturer, the director of the IoT and Robotics Education Lab, and a PhD student adviser. He is a co-author of ten books and more than 200 research papers.

Copyright © 2020 Inderscience Enterprises Ltd.

This paper is a revised and expanded version of a paper entitled 'A formal validation method for trustworthy services composition' presented at International Conference on Networking and Network Applications, Hakodate, Japan, 23 July 2016.

1 Introduction

Web services are often selected and combined to provide customers with all kinds of on-demand manufacturing services. Web services are also used in more areas, including business and mission critical applications. Thus, the service composition processes should also be concerned with the non-functional attributes of services, such as QoS, reputation, security, etc., and these attributes are important to whether the user's needs are met from all aspects. Therefore, in the system analysis and design phase, the establishment of web service composition model and the analysis of its functional correctness, the satisfiability of QoS, and other non-functional attributes, are also important methods to improve as the credibility (or trustworthiness) of the composed web services (Bucchiarone et al., 2007; Yun et al., 2013). Trust in a service is one of the most important factors when the web service is selected by the users (Ding et al., 2015; Habib et al., 2013). In this study, the web service credibility is the weighted sum of the non-functional attributes of the service, with the QoS, reputation, security and cost as the main factors.

Web service composition verification is the key to ensure that the services are implemented correctly. At present, there are three main types of service composition technologies. First, the workflow technology, which is implemented by defining process. This approach is also known as static composition, and it is supported by the tools such as workflow foundation and BPEL4WS. Second, semantics-based web services composition, which can realise the automatic interaction among web services by defining good semantic information and powerful semantic reasoning ability. Tools like DAML-S and OWL-S support this approach. Third, with the aid of the intelligent planning method of artificial intelligence, the service composition is transformed into the problem of intelligent search of the state space. Whatever type of service composition method is used, the support of the service composition language is needed, where commonly used are Workflow Foundation, BPEL4WS, WSFL, WSCI, OWL-S, etc. Visual IoT/robotics programming language environment (VIPLE) (Chen, 2018) is the latest composition language that can support both business composition and IoT/robotics applications (Chen and De Luca, 2016, 2017, 2018). Service composition language describes how to compose a number of basic services into a composed service. However, due to the lack of formal analysis capability, it is difficult when using these composition languages to analyse a service composition process model when it set up, and it is difficult to verify the correctness of service composition. To address these issues, a formal model of services composition can be used to determine whether the model keep the fidelity with the service, and whether the service is satisfied with the specific required attributes.

There are different ways of service modelling: convert the web service composition into Petri network, pi-calculus, and automaton (Bucchiarone et al., 2007). Then, the security, boundedness and deadlock of web service composition can be verified by using the verification methods for these models.

In order to verify the credibility of service composition and to express the dynamic changes of non-functional attributes of service in service composition process, this study applies the linear logic (LL) (Habib et al., 2013) and adds the non-functional attributes in the logical deduction rules composed with the pi-calculus. Then, the change of the credibility of the service can be fully observed in the formal analysis of the service composition process. Through a travel agent system case study, the proving process of the LL is converted into the corresponding pi-calculus expression, and the correctness of service composition scheme is verified by using VIPLE, which is developed based on pi-calculus (Chen and De Luca, 2018).

The rest of the paper is organised as follows. Section 2 presents the related work in the area. Section 3 introduces the LL deduction and its extension to include the non-functional attributes needed for this study. Through a travel agency case study, Section 4 studies the service composition verification using VIPLE and pi-calculus. A case study on service composition is given in Section 5. Section 6 concludes the paper.

2 Related work

Many studies existed that have used the formal verification of web service composition by using process algebra and pi-calculus (Yun et al., 2013; Chen and De Luca, 2016, 2017; Papapanagiotou and Fleuriot, 2011; Zhang et al., 2011). However, the research of web services by using process algebra has been focused on the functional modelling and correctness verification. The verification of non-functional attributes are not included, such as time, resource, power consumption, etc. Currently, there are some studies on modelling the non-functional attributes (Bucchiarone et al., 2007; Yun et al., 2013; Papapanagiotou and Fleuriot, 2011; Zhang et al., 2011; Wang et al., 2014; Xiao et al., 2012). The research of Xiao et al. (2012) extends the process algebra, marks the shifting cost of the process, marks the execution price of process action. They are based on the calculus communication system (CCS), and the cost and price are added into the process state shifting system PPA. According to the PPA, the price and the cost of the service have been changed when services shift and composite, and its validity is verified through examples. In

Zhang et al. (2011), the time constraints are added to the pi-calculus, and the success of the service is evaluated according to the response time. An example is given for demonstrating the deducing process. The study of Wang et al. (2014) is based on process algebra. The QoS attribute is used as an additional tag of action to identify the quality attributes when service executes. The related semantics are redefined in CCS, so that the QoS attribute values can be integrated when the service state shifts.

Although these studies of the formal verification of service composition have considered the non-functional attributes, they are simple extensions of the original theory, and they have not proved the soundness and correctness of the extended algebraic system. LL (Girard, 1987) is a refined version of the classical logic, and it provides a method for tracking resources. In LL, the inputs and outputs of the available services are considered to be the resource that is consumed and generated. We apply the proof-as-process (Abramsky, 1994) to make each step of the process of the composition correspond to an LL expression, so that the state shifting of the service behaviour is transformed into the corresponding LL expression (Miller and Pimentel, 1999), with a request for the collection of the service composition as a result. Once the proof is completed, the pi-calculus expression of the composed service is automatically abstracted from the proof. Furthermore, an expression of pi-calculus can be fully expressed as an expression in BPEL (Ferraa, 2004), so if it can prove the results by LL, the service composition scheme can be directly applied to the actual development.

Considering an LL proof as a pi-calculus process is first proposed by Abramsky (1994) and described in detail by Bellin and Scott (1994). The work in Bellin and Scott (1994) uses a style of type theory to express LL pi-calculus in the LL reasoning rules, such a service composition process model rendered by pi-calculus can be directly generated from the proof. Unlike other methods, applying the non-functional attributes is only the filtering of the resulting schemes. The study in Küngas and Matskin (2005) is based on partial deductive technique extended LL theory to prove and design a composition engine, in order to detect missing web services. At the same time, it is shown that the natural attributes of LL's resource awareness make the distinction between the resources, the statistics and even the dynamic update of the resources become possible. The functional and non-functional attributes of the service are converted into the proposition of the logical axioms, which are considered in the theory proving process in Rao et al. (2006). At the same time, the conversion relation between LL and pi-calculus is given, and LL rule can be converted into pi-calculus expression. However, the effect of non-functional attributes on LL is not considered.

Generally speaking, the studies on the formal verification of web service composition have made a great progress. However, the verification of the credibility of service composition is still insufficient. In this paper, we will perform both theoretical analysis as well as programmatic verification. We will use visual VIPLE for implementing the business program. VIPLE is based on pi-calculus and it can automatically convert any VIPLE workflow into pi-calculus expressions. A pi-calculus-based programming language called Pict was developed at the University of Edinburgh in 1992 (Pierce and Turner, 1997; Sewell et al., 2010). It is a text-based language, which is more difficult to use with the visual workflow process. The language is not in active development at this time. Another approach of modelling business process and workflow is the Petri net (Derni et al., 2019; Ghomri et al., 2018; Latorre-Biel et al., 2017). Petri nets have strong ability of modelling the flow of tokens and event-driven programs. However, Petri net is less powerful than pi-calculus in perform reasoning and proof. In our previous research, we conducted a theoretical study on embedding the non-functional features into the validation process of business process (Li et al., 2016). In this paper, we extend the theoretical work and apply it into actual service application and verification. We implement a case study using VIPLE environment.

3 LL deduction with non-functional attributes

3.1 LL representation of web services composition

Definition 1. LL grammars. The LL syntax used in this study is exemplified as follows:

$$A ::= P \mid A \multimap A \mid A \otimes A \mid A \oplus A \mid !A \mid 1$$

where P represents a proposition, A represents a propositional formula.

- Multiplicative conjunction ⊗: ⊗ represents that A and B are consumed or generated at the same time. If $A \otimes B$ represents resources, then both A and B have to be consumed at the same time. If $A \otimes B$ represents the needed result, then we can divide the resources into two parts r and r', where r can generate A, r' can generate B.
- Additive disjunction ⊕: If $A \oplus B$ is resources, if only one of A or B can be satisfied, the target can be achieved. If $A \oplus B$ is result, then means at least one of A and B is satisfied.
- Liner implication $\multimap$: $A \multimap B$ represents that only if the resource A is available, the target B can be achieved.
- Of course modality !: $!A$ represents resource A is infinite, that is A can be unlimitedly used.
- 1: represents the ultimate goal, and there is no resource request.

In terms of resource acquisition, logical expression $A \otimes B \multimap C \otimes D$ means to obtain resources C and D at the same time can only be achieved when A and B are obtained at the same time. After this formula is executed, if A and B are consumed, then C and D are produced. The expression of

$A \multimap: B \oplus C$, represents that if there is a resource A, you can get B or C, but don't know which one.

Process is considered to be a communication by differing a port or a channel, and named as a free variable. Proofs as processes (pi-calculus or CCS formula) is put forward by the Abramsky (1994), who made an in-depth research on the transformation from the theorem proving process in LL into process calculus. For example, multiplicative conjunction $\otimes$, additive disjunction $\oplus$, and unbounded operator !, respectively, correspond to composition, selection, copy operation of process calculus. The process calculus is used to describe the service flow of the software in many studies. So it is also reasonable to use LL to describe the web service flow.

According to Rao et al. (2006), a requirement of web service composition (including functional and non-functional attributes) is usually represented as a LL formula in (1):

$$(\Gamma_c, \Gamma_v); \Delta_c \vdash ((I \otimes P) \multimap (O \oplus F) \oplus E) \otimes \Delta_r \quad (1)$$

Formula (1) is divided into two parts, and bounded by $\vdash$, the left part is taken as resources that can be consumed, the right is the goal to reach or the task to complete. Among them, Γ_c and Γ_v are respectively available value increase and core service set, both expressed by the extended logic axiom. Δ_c is a non-functional constraint condition set, Δ_r represents a non-functional result set. $I \otimes P \multimap: (O \otimes F) \oplus E$ is a functional description of demand services. I and O are both a letter set of multiplicative conjunction with an of course modality. I represents an input parameter set of service, I represents an output parameter set of generating service, P and F are a set of multiplicative conjunction of preposition and result, respectively. E represents an exception set. The semantics of this formula can be understood as follows: given available atomic services and constraints condition set, find a satisfying composition service scheme that from the input I produces O and from state P shifts to F. An exception is thrown if the service fails.

LL can represent the system characteristics, which can be both functional attributes and non-functional attributes (Rao et al., 2006; Papapanagiotou and Fleuriot, 2011). In Bellin and Scott (1994), non-functional attributes of web services are classified into three categories: the attribute with quantitative consumption, such as: cost, price, time, etc.; the qualitative attribute, is the constraints specified by a web service for execution, such as calling a service requires security permissions. These two classes are the resources in the left side of the LL expression. The third class is the qualitative result, such as the type of service, service provider or geolocation, specifically refers to the results of the service context. In the LL expression, they are as the right side target. In this study, the web service attributes are expressed as QoS, credit, security and cost, and in LL, all of these can be expressed as the resource that can be consumed.

Different from classical logic, LL is a kind of computable, resource-oriented logic. In LL, the assumption is 'the premise is the resource'. All the assumptions in the proof must be consumed 'accurately once'. Therefore, the resources in the LL are 'generated' or 'consumed', rather than can be 'copied' or 'ignored' in the traditional logic, in short, the logical premise in LL will be the resources that can be consumed. In linear logical expressions, the resource labelled with ! can be used unlimitedly. In addition, LL is not about the authenticity of proposition, but about the calculation.

3.2 Logical deduction with non-functional attributes added

Proofs-as-processes theory allows us to perform LL proof and creating a web service composition with the pi-calculus. The process is to specify the available web service description to a LL formula, to construct a LL description of a service composition request, and to prove that it is the final result. Based on the available services, the proof of the description of the requested service $R$is demonstrated as follows. Assume $\{a_1, a_2, \ldots, a_k\}$ is the collection of the input parameters in I, $\{r_1, r_2, \ldots, r_k\}$ is the collection of the output parameters in O, $\{p_1, p_2, \ldots, p_k\}$ is the collection of the actions of the generation from the input parameters to the output parameters, and there is a path of service shifting from p_1 to p_k. Then, when all service behaviours obtain output results, this path is correct, namely, the service composition meets user needs. The formula using LL is as following:

$$\frac{\Gamma \vdash a_1 \multimap_{P_1} r_1, \ldots, a_k \multimap_{P_k} r_k}{\vdash !a_1 \otimes \cdots \otimes !a_k \multimap_{P_1 | \ldots | P_k} g_r}$$

where g_r represents the target output of the service composition.

Now, we consider satisfying both functional and non-functional attributes. The goal of this study is to compose service correctly after the selection of the composition objects (atomic services or composed services) to meet the user's needs. In the composition process, the non-functional attributes considered include QoS, credit, security, and cost.

In the following, the linear logical inference rules are constructed after adding the non-functional attributes by adding the description of the non-functional attributes to the pi-calculus.

Definition 2. The definition of the non-functional attributes of the sum operations, and additive disjunction in the pi-calculus expression are as follows:

1 Sum operation

$$SUM: \frac{P \to (\alpha)P'}{P + Q \to (\alpha)P'}$$

where *SUM* operation represents: process P after the completion of the operation α evolves into a process P', then process $P + Q$ can also turn into process P' after the completion of the operation of α. This is due to the

process of $P + Q$ can select P between P and Q to execute. After the sum operation, the cost (*Cost*), reputation (*Rep*), security (*Sec*), QoS value of service are the parameters for the selection of P or Q, as follows:

$$Cost(P'+Q') = Cost(P) \text{ or } Cost(Q)$$

$$Rep(P'+Q') = Rep(P) \text{ or } Rep(Q)$$

$$Sec(P'+Q') = Sec(P) \text{ or } Sec(Q)$$

$$QoS(P'+Q') = QoS(P) \text{ or } QoS(Q)$$

Corresponding to the LL and the additive disjunction, the multiplicative conjunction of parameters are as follows:

$$Cost(P'+Q') = Cost(P) \ \oplus \ Cost(Q)$$

$$Rep(P'+Q') = Rep(P) \ \oplus \ Rep(Q)$$

$$Sec(P'+Q') = Sec(P) \ \oplus \ Sec(Q)$$

$$QoS(P'+Q') = QoS(P) \ \oplus \ QoS(Q)$$

2 Additive disjunction

$$COM: \frac{P \to (\alpha)P'Q \to (\alpha')Q'}{P|Q \to (\alpha, \alpha')P'|Q'}$$

Additive disjunction represents: processes P and Q, respectively, to execute α and action α' then evolve into P' and Q', thus, the concurrent composition P|Q of processes P and Q, also can evolve into $P'|Q'$.

After the additive disjunction, the cost (Cost), reputation (Rep), security (Sec), the change of QoS of service are as follows:

$$Cost(P' \mid Q') = Cost(P) + Cost(Q)$$

$$Rep(P' \mid Q') = \big(Rep(P) + Rep(Q)\big)/2$$

$$Sec(P' \mid Q') = \min\{Sec(P), Sec(Q)\}$$

$$QoS(P' \mid Q') = QoS(P) \times QoS(Q)$$

Corresponding to the multiplicative conjunction in the LL, the additive disjunctions of the parameters are expressed as:

$$Cost(P' \mid Q') = Cost(P) \otimes Cost(Q)$$

$$Rep(P' \mid Q') = Rep(P) \otimes Rep(Q)$$

$$Sec(P' \mid Q') = Sec(P) \otimes Sec(Q)$$

$$QoS(P' \mid Q') = QoS(P) \otimes QoS(Q)$$

According to the logic that LL is computable, resource-oriented, based on Table 2 in Rao et al. (2006), the quantitative resource *Con* that can be consumed is added. Based on Definition 2, the modified inference rules are shown in Figure 1.

In Figure 1, Con^a and Con^b are two resources that can be consumed. After applying the additive disjunction and multiplicative conjunction in the LL, the calculation results are $Con^a \otimes Con^b$ and $Con^a \oplus Con^b$.

Figure 1 A linear logical inference rule with non-functional attributes embed

Logical axiom and Cut rule:

$$A(x)\text{|-}0 : A(x) \ \ (ID) \qquad \frac{Con^a, \Gamma\text{|-}P : A(a_1) \ \ Con^b, \Gamma', A(a_2)\text{|-}Q : C}{(Con^a \otimes Con^b), \Gamma, \Gamma'\text{|-}(P.\overline{a_1}a_2.Q) : C} \qquad (Cut)$$

Rules for propositional constants

$$|\text{-}1 \quad \frac{Con^a, \Gamma, A\text{|-}A}{Con^a, \Gamma, 1\text{|-}A}$$

$$\frac{Con^a, \Gamma, A(a), B(b)\text{|-}P : C}{Con^a, \Gamma, (A \otimes B)(a,b)\text{|-}P : C} \ (L\otimes) \qquad \frac{Con^a, \Gamma\text{|-}P : \mathrm{A}(a) \ \ Con^b, \Gamma'\text{|-}Q : B(b)}{(Con^a \otimes Con^b), \Gamma, \Gamma'\text{|-}(P \mid Q) : (A \otimes B)(a,b)} \ (R\otimes)$$

$$\frac{Con^a, \Gamma\text{|-}A(a) \multimap_P B(b)}{Con^a, \Gamma, A(a)\text{|-}P : B(b)} \ (Shift) \qquad \frac{Con^a, \Gamma, A(a)\text{|-}P : B(b)}{Con^a, \Gamma\text{|-}A(a) \multimap_{a.P.\bar{b}} B} \ (R\multimap)$$

$$\frac{Con^a, \Gamma, A(a)\text{|-}P : C(c_1) \ \ Con^b, \Gamma, B(b)\text{|-}Q : C(c_2)}{(Con^a \oplus Con^b), \Gamma, (A \oplus B)(a+b)\text{|-}(P+Q) : C(c_1+c_2)} \ (L\oplus) \qquad \frac{Con^a, \Gamma\text{|-}P : A(a)}{Con^a, \Gamma\text{|-}P : (A \oplus B)(a)} \ (R\oplus)$$

Rules for exponential!:

$$\frac{Con^a, \Gamma\text{|-}\Delta}{Con^a, \Gamma, !A(0)\text{|-}\Delta} \ (W!) \qquad \frac{Con^a, \Gamma, A(a)\text{|-}\Delta}{Con^a, \Gamma, !A(a)\text{|-}\Delta} \ (L!) \qquad \frac{Con^a, \Gamma, !A(a), !A(a)\text{|-}\Delta}{Con^a, \Gamma, !A(a)\text{|-}\Delta} \ (C!)$$

Structural congruence for a process calculus

$$P \mid Q \equiv Q \mid P \quad P+Q \equiv Q+P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad (P+Q)+R \equiv P+(Q+R)$$

$$P \equiv P \mid 0 \equiv P+0 \equiv P.0 \equiv 0.P$$

$$A(a) \otimes B(b) \equiv A \otimes B(a,b)$$

$$\overline{(a_1, a_2, ..., a_n)}(b_1, b_2, ..., b_n) \equiv (\overline{a_1}b_1, \overline{a_2}b_2, ..., \overline{a_n}b_n)$$

4 Service composition in VIPLE and pi-calculus

In this section, we present VIPLE workflow and its pi-calculus conversion in the service composition.

4.1 Introduction to VIPLE

VIPLE is a workflow-based programming environment (Chen, 2018; Chen and De Luca, 2016, 2017). It has an open interface in JSON, which allows for other services and devices to communicate with it, as show in Figure 2. VIPLE consists of four major components:

1 a visual programming interface for the developers to draw (drag and drop) the application diagrams

2 a repository of built-in activities and services to simplify the development jobs

3 a compiler that translates the visual code/diagram into the executable

4 the runtime that executes the code and communicates with the RaaS units through HTTP and TCP.

The RaaS units could be simulators and physical devices. The data interfaces between VIPLE and the RaaS units are defined in JSON (JavaScript Object Notation) format, which plays a vital role in the platform independence of the environment. Any simulator and physical device can be added into the system, as long as they can provide a TCP socket or a web socket for the communication and can encode and decode the JSON object for processing the required actions. The usability of a language largely depends on the availability of library functions or called services. Figure 3 shows the three sets of ASU VIPLE services.

The first set contains the basic activities that forms the essential construct of a programming language, such as data, variable, calculate, if, switch, and while. The join and merge activities allow the user to form parallel workflows. The custom activity is used for forming a component with workflow.

Figure 2 VIPLE architecture and its front end services and devices (see online version for colours)

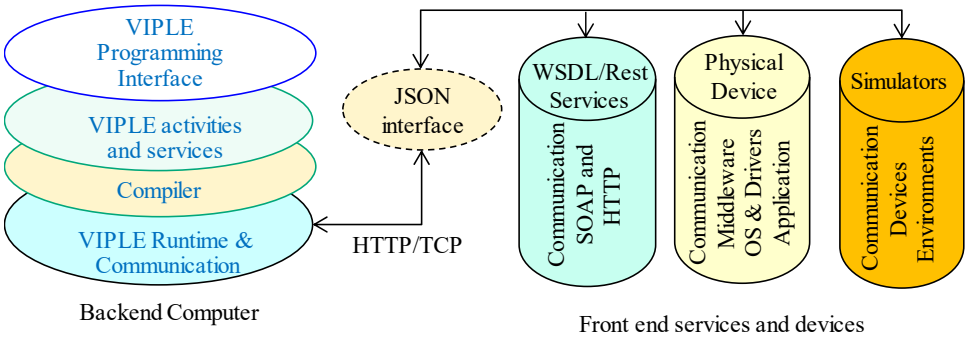
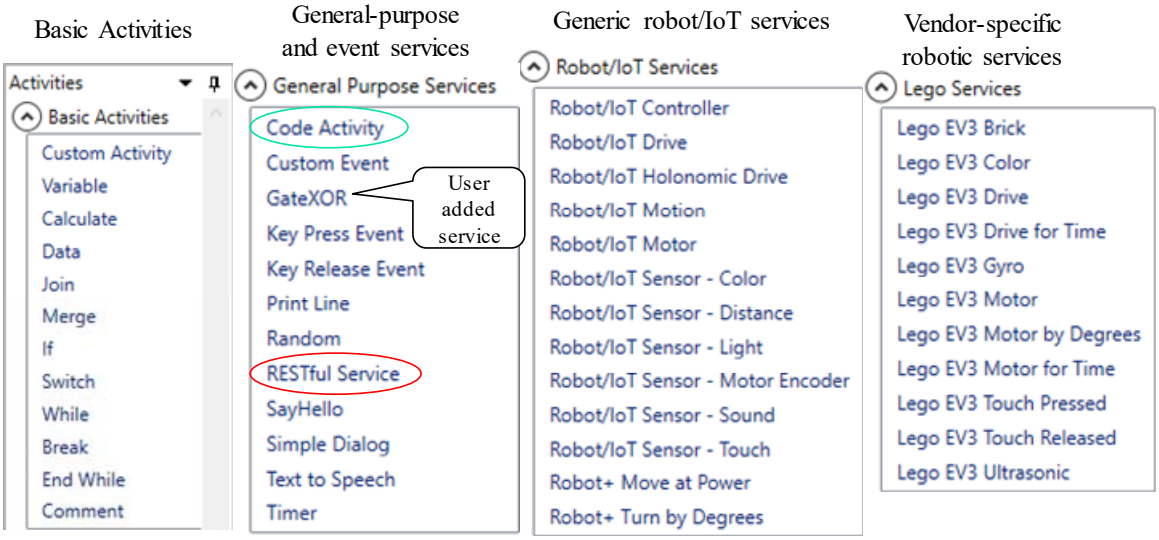


Figure 3 VIPLE services (see online version for colours)



The second set contains the general services, including input/output services (simple dialog, print line, text to speech, and random), event services (key press event, key release event, custom event, and timer), and RESTful services. WSDL services can be included through the menu command services → Add WSDL Service. Using these set of the services, we can implement general workflow applications such as the travel agency application to be discussed the case study in the next section. Code activity in this set of service allows us to write text-based code. In other words, we could use code activity services to draw flowchart, resulting an architecture-driven software development style.

The third set is the generic robotic services. VIPLE offers a set of standard communication interfaces, including Wi-Fi, TCP, Bluetooth, USB, local host, and WebSocket interfaces. The data format between VIPLE and the IoT/Robotic devices is defined as a standard JSON (JavaScript Object Notation) object. Any robot that can be programmed to support one of the communication types and can process the JSON object can communicate with VIPLE and be programmed in VIPLE. As shown in the second part of Figure 3, all VIPLE services that start with 'robot' are generic robotic services. We can use these services to program our simulated robots and custom-built physical robots.

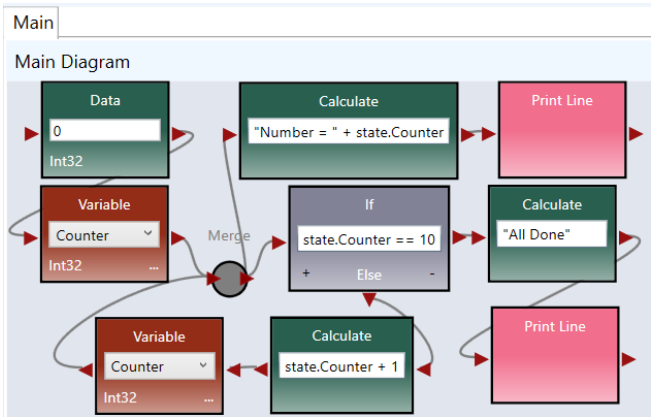
The fourth set is the vendor-specific services. Some robots, such as LEGO robots and iRobots, do not offer an open communication and programming interfaces. In this case, we can offer built-in services in VIPLE to access these robots without requiring any programming efforts on the device side.

VIPLE is available for free downloading at: http://neptune.fulton.ad.asu.edu/VIPLE/.

4.2 *VIPLE Diagram and pi-calculus*

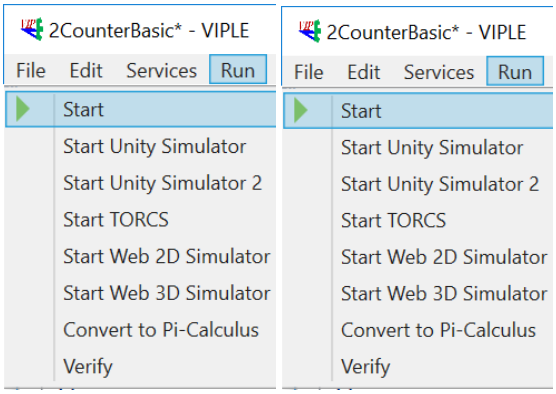
We start with a simple VIPLE program that increases a counter variable from 0 to 10. Figure 4 shows the VIPLE diagram.

Figure 4 VIPLE diagram for counting from 0 to 10 (see online version for colours)



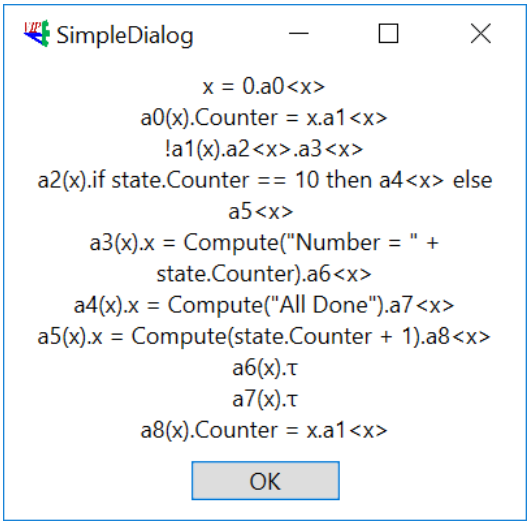
In the diagram, we use a data activity to introduce a constant value 0. The value is assigned to the variable counter as its initial value. The counter value goes through a merge activity to the If activity, which checks if the counter value. The program stops if the counter value is 10. If the counter value is not 10, it uses a calculate activity to add 1 to the counter variable and then write the value back to the variable. Then, the variable goes back the merge activity. The merge activity requires on incoming value to proceed. For the first time, the merge's input value comes from the variable's initial value and then, the value comes from the loop. We use print line service to print the values. We use VIPLE menu run – start to execute the program and the print out shown in Figure 5.

Figure 5 Start menu and output (see online version for colours)



Using the menu command convert to pi-calculus, we can convert the any VIPLE diagram into pi-calculus expressions. Figure 6 show the pi-calculus expressions from the diagram in Figure 4.

Figure 6 Pi-calculus expressions of Figure 4 diagram (see online version for colours)



In the case study in the next section, we will use VIPLE and pi-calculus to verify a more complex example.

5 Service composition case study

In this section, we use a travel booking system as an example to illustrate the process.

Figure 7 Workflow diagram of the travel agent system (see online version for colours)

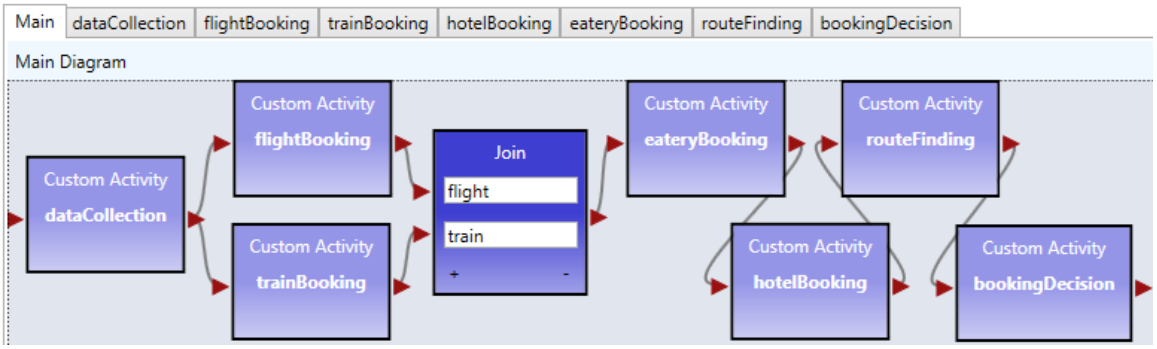


Figure 8 Converting VIPLE diagram into pi-calculus expressions automatically (see online version for colours)

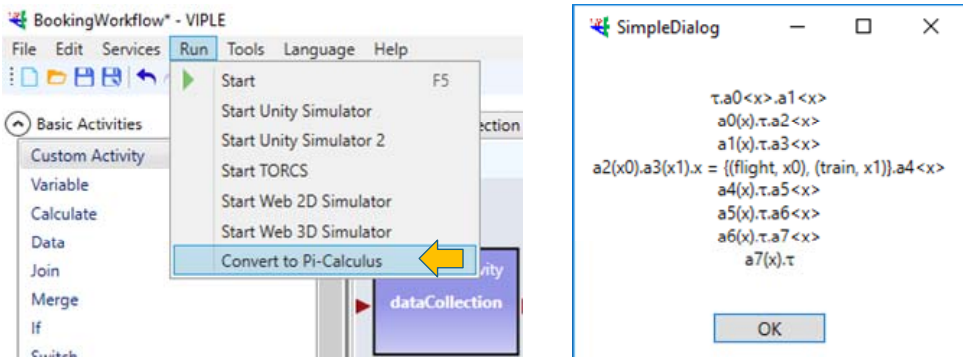
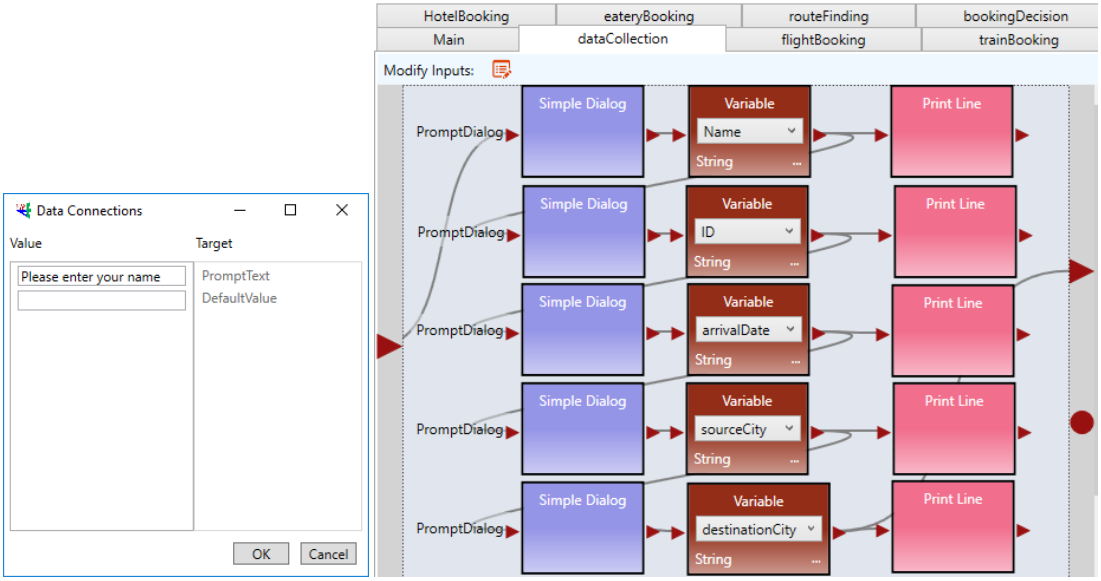


Figure 9 Implementation of dataCollection service through SimpleDialog service (see online version for colours)



A simplified service composition workflow is shown in Figure 7. It shows two alternatives starting with flight booking or train booking, followed by hotel and restaurant booking. A path from Start to End defines a route query. In these services, the 'route query' service does not need cost, each of the others has the lowest cost control. Assume that it does not take into account the internal operation of each service, and take each service as an existing service. There are five services in the travel agency system: flightBooking, trainBooking, hotelBooking, eateryBooking, and routeFinding.

VIPLE diagram is based on pi-calculus, and the entire diagram can be translated into pi-calculus expressions automatically. First, we choose menu command Run, and then choose convert to pi-calculus. The corresponding pi-calculus expressions are generated, as shown in Figure 8.

According to the composition process of travel agency services, each block (custom activity) needs to be further implemented as a service (component), as shown in the top banner of the workflow diagram. These services include:

1 Date input and store: dataCollection

This service collect the information needed by all the booking services, including: personal identification PI, arrival date AD (returning date RD, optional), start city SC, destination city DC, start date SD and other information, such as maximum costs for hotel and restaurant, then outputs plane ticket (PT, plane ticket) and the arrival date (AD, arrival date).

For this service, we use a sequence of SimpleDialog services, which allow the user to enter the travel information. The left part of Figure 9 shows the configuration of a SimpleDialog service, which shows prompt text and a textbox for data input. The entered data is stored in a variable. The variables can be accessed by other services in the Main diagram.

The right part of Figure 9 shows the entire dataCollection with a sequence of SimpleDialog services and variables, as well as the input and output connection of the service.

The service is executable. As an example, Figure 10 shows a few input steps and the printout on the console window after all the data are entered.

2 Plane ticket booking service: flightBooking

According to the travel information AD, RD, SC, DC, and maximum cost Cost1 to query the available flights and ticket prices from different airlines. It outputs the available flights and ticket prices from different airlines.

To query the fight information and prices, we can use flight information web services, such as: http://ws.51book.com:8000/ltips/services/getAvailableFlightWithPriceAndCommisionService1.0?wsdl.

Figure 11 shows the availability of the web service though Visual Studio WCF service test tool. It can invoke the service and obtain the results.

In VIPLE, we can add a WSDL web service into the diagram by choosing the menu item 'Services → Add WSDL Service', as shown in Figure 12. The dialog box will pop up, asking for enter a WSDL service address. Once the address is entered, the service operations will be added into the general service list in Figure 3. Then, the service operations will be available for dragging and dropping into the application diagram.

In the way, the following train, hotel, and eatery services can be added into the diagram and then used in each activity.

3 Train ticket booking service: trainBooking

According to the travel information DD, RD, SC, DC, and maximum cost Cost2 to query the available trains and ticket prices. It outputs the available trains and ticket prices. The service can be implemented similar to flightBooking service by calling train booking web services.

4 Restaurant booking service: eateryBooking

According to the user destination city DC, arrive date AD, books a restaurant meeting the user's willingness to pay the cost of the Cost3 at most, then outputs the available and qualified restaurants (EO, Eatery Order). Information such as the location of the restaurant is included in the order. The service can be implemented similar to flightBooking service by calling restaurant web services.

5 Hotel booking service: hotelBooking

According to the user destination city DC, arrival date AD, return date RD books a hotel meeting the user's willingness to pay the cost of the Cost4 at most, then outputs the order of booked hotel (HO, Hotel Order). Information such as the location of the hotel is included in the order. The service can be implemented similar to flightBooking service by calling hotel web services.

6 Route query service: routeFinding

According to the user destination city DC and order information (EO and HO), finds the route to the restaurants and hotels ROE (route of the eatery) and ROH (route of hotel). The service can be implemented similar to flightBooking service by calling map web services.

7 Booking decision service: bookingDecision

Based on the results from all the other proceeding services, this service will make the following decisions and final booking: First, it decides to book flight or train. Then, decide which flight or which train to book. For each of the other service outputs, it decides which hotel and which restaurant to book. The outputs of the services include reservation numbers and details of each service.

Each of these services includes QoS, security, reputation and cost. In fact, flight ticket booking service and train ticket booking service are all composed of many atomic services. For example, the train ticket booking service, at least, needs to query the ticket, if find the needed ticket, then pay the fare, the train ticket will be generated after the success of paying.

In a complete web service market environment, each service has more than one provider. According to the user's QoS, security, reputation and cost, calculate the credibility of each service, then add services meeting the lowest needs of users on the QoS, security, reputation and cost to each service class. This ensures that all compositions objects in the process of service composition are credible services.

Figure 10 The execution process of dataCollection service (see online version for colours)

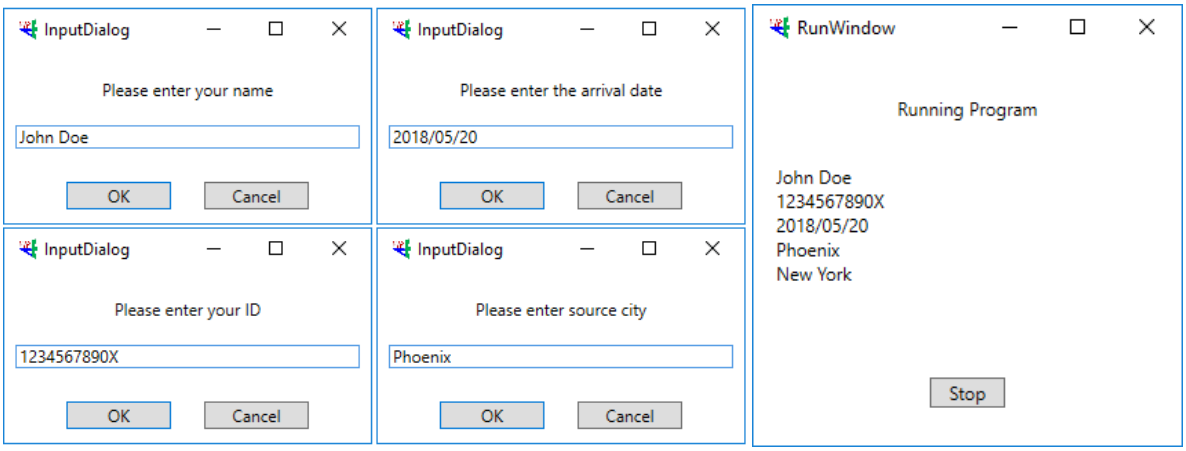


Figure 11 Testing the availability of the flight information web service (see online version for colours)

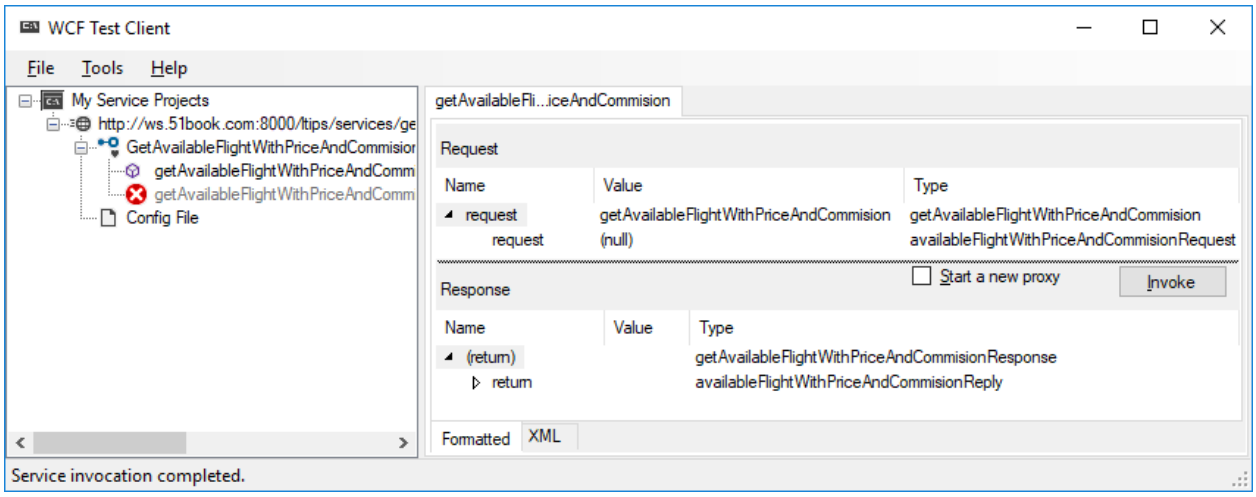
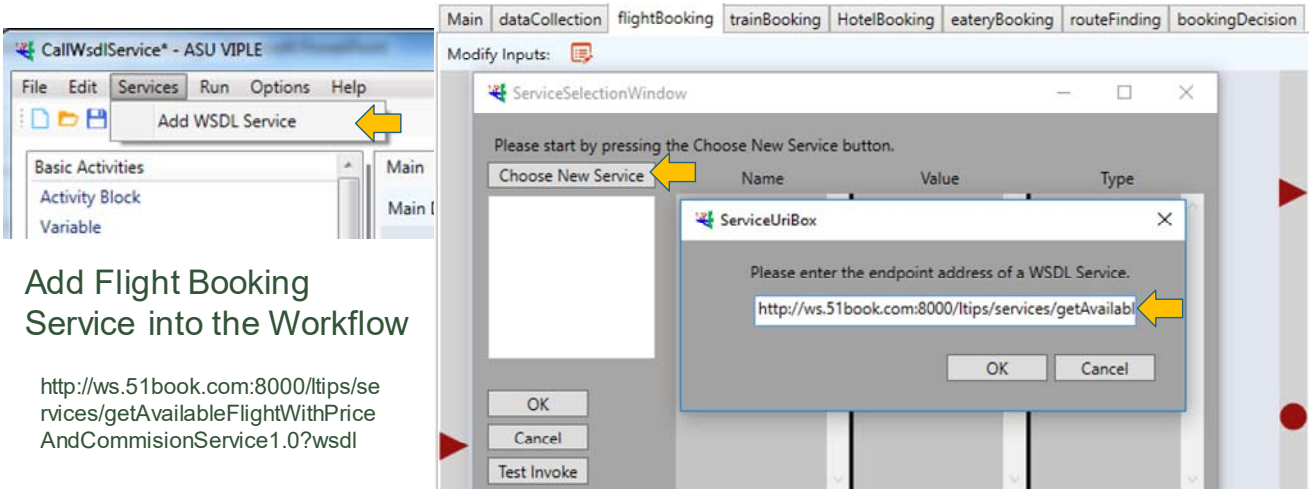


Figure 12 Add flight booking service into VIPLE diagram (see online version for colours)



5.1 Credibility deduction of composed services

Because in the composition of this case, there are both atomic and composed services, in the service composition, the social surplus maximisation is the composite objective. According to LL and service description, the system's demand results are shown in the formula (2).

$$\vdash !PI \otimes !SC \otimes !DC \otimes SD \multimap_{(flightBooking+trainBooking).(eateryBooking|hotelBooking).routeFinding} (PT \oplus TT) \otimes ROE \otimes ROH \quad (2)$$

Figure 13 LL reasoning process of travel agent system

$$\frac{\dfrac{Cost_{\square}^{c1} \mid - !PI \otimes !SC \otimes !DC \otimes SD \multimap_{flightBooking} PT \otimes AD}{!PI \otimes !SC \otimes !DC \otimes SD; Cost_{\square}^{c1} \mid - flightBooking: PT \otimes AD}(shift) \quad \dfrac{Cost_{\square}^{c2} \mid - !PI \otimes !SC \otimes !DC \otimes SD \multimap_{trainBooking} TT \otimes AD}{!PI \otimes !SC \otimes !DC \otimes SD; Cost_{\square}^{c2} \mid - trainBooking: TT \otimes AD}(shift)}{!PI \otimes !SC \otimes !DC \otimes SD; (Cost_{\square}^{c1} \oplus Cost_{\square}^{c2}) \mid - (flightBooking + trainBooking): (PT \oplus TT) \otimes AD}(L \oplus)$$

$$\frac{}{!PI \otimes !SC \otimes !DC \otimes SD; (Cost_{\square}^{c1} \oplus Cost_{\square}^{c2}) \mid - (flightBooking + trainBooking): (PT \oplus TT) \otimes !DC \otimes !AD}(W!) \quad \text{(S1)}$$

……

$$\frac{\dfrac{Cost_{\square}^{c3} \mid - !DC \otimes !AD \multimap eateryBooking^{EO}}{!DC \otimes !AD; Cost_{\square}^{c3} \mid - eateryBooking: EO}(shift) \quad \dfrac{Cost_{\square}^{c4} \mid - !DC \otimes !AD \multimap hotelBooking^{HO}}{!DC \otimes !AD; Cost_{\square}^{c4} \mid - hotelBooking: HO}(shift)}{!DC \otimes !AD; (Cost_{\square}^{c3} \otimes Cost_{\square}^{c4}) \mid - (eateryBooking \mid hotelBooking): EO \otimes HO}(R \otimes) \quad \text{(S2)}$$

……

$$\frac{\dfrac{\mid - !DC \otimes EO \multimap routeFinding^{ROE}}{!DC \otimes EO \mid - routeFinding: ROE}(shift) \quad \dfrac{\mid - !DC \otimes HO \multimap routeFinding^{ROH}}{!DC \otimes HO \mid - routeFinding: ROH}(shift)}{!DC \otimes EO \otimes HO \mid - routeFinding: ROE \otimes ROH}(R \otimes)$$

Cut with **(2)**

$$!DC \otimes !AD; (Cost_{\square}^{c3} \otimes Cost_{\square}^{c4}) \mid - (eateryBooking \mid hotelBooking).routeFinding: ROE \otimes ROH \quad \text{(S3)}$$

Cut with **(1)**

$$!PI \otimes !SC \otimes !DC \otimes SD; (Cost_{\square}^{c1} \oplus Cost_{\square}^{c2}) \otimes (Cost_{\square}^{c3} \otimes Cost_{\square}^{c4}) \mid - (flightBooking \oplus trainBooking).(eateryBooking \mid hotelBooking).routeFinding: (PT \oplus TT) \otimes ROE \otimes ROH \quad \text{(S4)}$$

$$\frac{}{(Cost_{\square}^{c1} \oplus Cost_{\square}^{c2}) \otimes (Cost_{\square}^{c3} \otimes Cost_{\square}^{c4}) \mid - !PI \otimes !SC \otimes !DC \otimes SD \multimap_{(flightBooking+trainBooking).(eateryBooking \mid hotelBooking).routeFinding} (PT \oplus TT) \otimes ROE \otimes ROH}(R \multimap)$$

Congruence

$$(Cost_{\square}^{c1} \oplus Cost_{\square}^{c2}) \otimes Cost_{\square}^{c3} \otimes Cost_{\square}^{c4} \mid - !PI \otimes !SC \otimes !DC \otimes SD \multimap_{(flightBooking+trainBooking).(eateryBooking \mid hotelBooking).routeFinding} (PT \oplus TT) \otimes ROE \otimes ROH \quad \text{(S5)}$$

According to the deduction rule of Figure 1, the LL reasoning process of the travel agent system can be given. In order to further simplify the reasoning process, the credibility of the service is represented by the cost of services, as shown in Figure 13. Because of the space limitation, the channel value of each parameter embedded in the pi-calculus process is not written.

In the reasoning process in Figure 13, first data collection can be needed step (S1) is to choose the plane ticket booking or the train ticket booking, then output the destination. Step (S2) is to choose a restaurant and hotel according to the destination. Step (S3) is to take the restaurant and hotel information as input information, and query routes to the restaurant and hotel. Step (S4) is a composition of four services, namely, plane or train ticket booking, restaurant booking, hotel booking, and route query.

The above reasoning processes have not added the possible exception. Each booking result may give an exception, where we assume that all bookings can be successful. In fact, if it cannot book successfully this time, you can redo the booking; there will always be a result, so the exception is not considered in the process of the composition.

After the above deduction, the expression in the last step (S5) is the result for the needs of users. The calculus of non-functional attribute Cost is added in the deduction, if $(Cost^{c1} \oplus Cost^{c2}) \otimes Cost^{c3} \oplus Cost^{c4}$ returns a smaller result than the total budget. Then, it gives $(PT \oplus TT) \otimes ROE \otimes ROH$, otherwise the composition fails. Here the result of the *Cost* is one of the two cases: $Cost^{c1} + Cost^{c3} + Cost^{c4}$ or $Cost^{c2} + Cost^{c3} + Cost^{c4}$.

Rewriting $(PT \otimes AD) \oplus (TT \otimes AD)$ to $(PT \oplus AD) \otimes AD$, $(PT \otimes AD) \oplus (TT \otimes AD) = (PT \oplus TT) \otimes AD$ will not affect the meaning of the result.

In the reasoning in Figure 13, it just adds a non-functional attribute Cost into the proving process, after the composition, the reputation and security can also be expressed in the form of a resource with the same proving process, thus not describe them here. Therefore, in the process of reasoning, because of the addition of non-functional attributes, in the evolution process, it is capable to compare and analyse the cost, reputation and security, and in each step to produce an optimal result, ensuring the credibility of the composed service.

5.2 Expression and reasoning of pi-calculus for composed services

In order to clarify the reasoning process in steps, the proving process of the LL in Figure 13 is converted into the corresponding pi-calculus expression. Because of the complexity of pi-calculus, it is difficult to realise the direct simulation of its grammar. The VIPLE developed by Chen (2018) and Chen and De Luca (2016, 2017) can graphically describe the activities in the pi-calculus, as well as a step by step interactive process of operation. VIPLE is able to demonstrate the execution process of composed services, clearly and directly show the proposed results, without the need to further study the complex grammar of pi-calculus.

However, since VIPLE can only represent the pi-calculus expression with functional attributes, in the deduction process, the non-functional attribute of cost is removed after conversion into a pi-calculus expression.

In order to convert the LL expression into the pi-calculus expression, the Rao et al. (2006) is used to construct the following transformation rules:

If A, B are the input resources of the right of the LL expression, then:

$$A\overline{a} < \ldots).0$$

$$A \otimes B(va,b)\left(\overline{z} < a, b > .\left(\overline{a} < \ldots > .0 \,\|\, \overline{b} < \ldots > .0\right)\right)$$

$$A \oplus B(va, b)\left(z(u, v).(\overline{u} < x > .x < \ldots > .0 + v < y > .y < \ldots > .0)\right)$$

If both A and B are the input resource S of the left of the LL expression, then:

$$A \;\; a(\ldots).0$$

$$A \otimes B \; z\,(a, b).\left(a\,(\ldots).0 \,\|\, .0 \,\|\, b(\ldots).0\right)$$

$$A \oplus B\,(v\,a, b)\left(z < u, v > .\left(u\,(x).x(\ldots).0 + v(y).y(\ldots).0\right)\right)$$

In order to facilitate writing, the formulae below adopt the abbreviations of seven services (processes) DataCollection: flightBooking, trainBooking, eateryBooking, hotelBooking, routeFinding, bookingDecision which are respectively PlB, TrB, EaB, HoB, and RoF, and the converted formulae are:

$$PlB(ppi,\ pdc,\ psd,\ psc,\ pc,\ pt,\ ad)$$
$$= ppi(pi).pdc(dc).psd(sd).psc(sc).\left(vz_1,\ z_2\right)$$
$$\left(\overline{p_c} < z_1, z_2 >, z_1 < pt > .\overline{z_2} < ad > .0\right)$$

$$TrB(tpi, tdc, tsd, tsc, tc, tt, ad)$$
$$= tpi(pi).tdc(dc).tsd(sd).tsc(sc).\left(vz_3,\ z_4\right)$$
$$\left(\overline{t_c} < z_3, z_4 >, \overline{z_3} < tt > .\overline{z_4} < ad > .0\right)$$

$$Eab(edc, ead, eeo, eo) = edc(dc).ead(ad).eeo < eo > .0$$

$$HoB(hdc, had, hha, ho) = hdc(dc).had(ad).hho < ho > .0$$

$$RoF(rdc, reo, rho, rroe, rroh, roe, roh)$$
$$= z_5(a, b)(vrdc, reo)\left(a\left(rdc(dc).reo(eo)\right).0 \,\|\right.$$
$$\left.(vrdc, rho)\left(b\left(rdc(dc).rho(ho)\right).0\right).\right.$$

$$(vrroe, rroh)\left(\overline{z_6}(rroe, rroh).\left(\overline{rroe} < roe > .0 \,\|\, \overline{rroh} < roh > .0\right)\right)$$

The service composition of the requirements is represented by the formula (3).

$$\begin{aligned} &|\!- !PI \otimes !SC \otimes !DC \\ &\otimes SD \multimap_{(flightBooking).(eateryBooking|hotelBooking).routeFinding} \\ &(PT \oplus TT) \otimes ROE \otimes ROH \end{aligned} \tag{3}$$

Use shift rule in Figure 1, we can obtain:

$$!PI \otimes !SC \otimes !DC \otimes SD \,|\!- P : (PT \oplus TT) \otimes ROE \otimes ROH,$$

where P is expressed as follows:

$$P \equiv (flightBooking + trainBooking)$$
$$.(eateryBooking\,|hotelBooking).routeFinding$$

Set the input in requirement composition as the target output, the output as the input resource, giving the dual of the above composition, which is converted into pi-calculus expression as follows:

$$Request(rpi, rdc, rsd, rsc,rc, rpt, rtt, roe, roh)$$
$$= \overline{rpi} < pi > .\overline{rdc} < dc > .\overline{rsd} < sd > .\overline{rsc} < sc > .$$

$$z_7(c, d)(c(z_8(a, b)(a(\overline{rc} < rpt, rtt > .(rpt(x).x(pt).0$$
$$+ rtt(y).y(tt).0)).0 \,\|\, b(roe).0))) \,\|\, d(roh).0))$$

The VIPLE execution process is not only a description of the service functions, but also an execution engine. The correctness of the composition scheme is described by the deduction process. A prover is currently being developed to prove the correctness of the pi-calculus expressions generated from VIPLE converter. Once this work is completed, we can perform automated proof of composite services described in this paper.

We also performing case studies of other types of composite services application in healthcare and in electronic commerce.

6 Conclusions

For composed services, in addition to the functional requirements that must be met, we should also consider the needs of the non-functional attributes, so that each step of the composition process could be verified. LL can carry non-functional attributes in the process of deduction, which is suitable for service composition verification. At the same time, the expression of pi-calculus can be easily converted into workflow diagrams in VIPLE. VIPLE diagrams are executable and can be converted into pi-calculus. Therefore, the service composition is verified by the combination of LL and pi-calculus in this study. Because of the grammatical complexity of pi-calculus, it is difficult to simulate. In order to ensure the correctness of the service composition process deduction, the VIPLE is used to represent the process of the calculation graphically. As the service credibility is closely related to the non-functional attributes, by adding the non-functional attributes to the LL deduction rules, the effect of the non-functional attributes on the credibility of the composed service is derived and verified.

Acknowledgements

This research is supported by Key Program of Visiting Studies for Excellent Youth Talents in Universities (No. gxfxZD2016250), Program for Science and Technology Innovative Research Team of Chuzhou University (Key Technologies and Applications of IoT), the International S&T Cooperation Program of Anhui Province (No. 1704e1002217).

References

Abramsky, S. (1994) 'Proofs as processes', *Theoretical Computer Science*, Vol. 135, No. 1, pp.5–9.

Bellin, G. and Scott, P.J. (1994) 'On the pi-calculus and linear logic', *Theoretical Computer Science*, Vol. 135, No. 1, pp.11–65.

Bucchiarone, A., Gnesi, S. and Beek, M.H.T. (2007) 'Formal methods for service composition', *Annals of Mathematics, Computing & Teleinformatics*, Vol. 1, No. 5, pp.1–10.

Chen, Y. (2018) *Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services*, 6th ed., Kendall Hunt Publishing, Dubuque, IA, USA.

Chen, Y. and De Luca. (2016) 'VIPLE: visual iot/robotics programming language environment for computer science education', *IPDPS Workshops*, pp.963–971.

Chen, Y. and De Luca, G. (2017) 'Visual IoT/robotics programming language in pi-calculus', *IEEE 13th International Symposium on Autonomous Decentralized Systems (ISADS)*, Thailand, pp.23–30.

Chen, Y. and De Luca, G. (2018) 'Technologies for developing a smart city in computational thinking', *International Journal of Simulation and Process Modelling*, Vol. 13, No. 2, pp.91–101.

Derni, O., Boufera, F. and Khelfi, M.F. (2019) 'Coloured Petri net for modelling and improving emergency department based on the simulation model', *International Journal of Simulation and Process Modelling*, Vol. 14, No. 1, pp.72–86.

Ding, Y., Wang, H.M., Shi, P.C. et al. (2015) 'Trusted cloud service', *Chinese Journal of Computer*, Vol. 38, No. 1, pp.133–149.

Ferraa, A. (2004) 'Web services: a process algebra approach', *Proc: ICSOC'04*, ACM Press, New York, USA, pp.242–251.

Ghomri, L. and Alla, H. (2018) 'Continuous Petri nets and hybrid automata: two bisimilar models for the simulation of positive systems', *International Journal of Simulation and Process Modelling*, Vol. 13, No. 1, pp.24–34.

Girard, J.Y. (1987) 'Linear logic', *Theoretical Computer Science*, Vol. 50, No. 1, pp.1–102.

Habib, S.M., Varadharajan, V., Mühlhäuser, M. et al. (2013) 'A framework for evaluating trust of service providers in cloud marketplaces', *Proceedings of the ACM Symposium on Applied Computing*, pp.-–965, *28th Annual ACM Symposium on Applied Computing*, SAC.

Küngas, P. and Matskin, M. (2005) 'Detection of missing web services: the partial deduction approach', *International Journal of Web Services Practices*, Vol. 1, Nos. 1–2, pp.133–141

Latorre-Biel, J.I., Jiménez-Macías, E., García-Alcaraz, J.L. et al. (2017) 'Modular construction of compact Petri net models', *International Journal of Simulation and Process Modelling*, Vol. 12, No. 6, pp.515–524.

Li, Y.M.; Zhao, S.H., Diao, H.L. and Chen, H.B. (2016) 'A formal validation method for trustworthy services composition', *International Conference on Networking and Network Applications*, Hakodate, Japan, July, pp.433–437.

Miller, D. and Pimentel, E. (1999) 'Linear logic as a framework for specifying sequent calculus', *Proceeding of the Annual European Summer Meeting of the Association for Symbolic Logic. Lecture Notes in Logic 17, Logic Colloquium'99*.

Papapanagiotou, P. and Fleuriot, J. (2011) 'Formal verification of web services composition using pi calculus and linear logic and the pi-calculus', *Ninth IEEE European Conference on Web Services*, IEEE, pp.31–38.

Papapanagiotou, P. and Fleuriot, J.D. (2011) 'A theorem proving framework for the formal verification of web services composition', *Proceedings of WWV 2011. EPTCS 61*, pp.1–16.

Pierce, B.C. and Turner, D.N. (1997) *Pict: A Programming Language Based on the Pi-Calculus*, Technical report, Computer Science Department, Indiana University.

Rao, J.; Küngas, P. and Matskin, M. (2006) 'composition of semantic web services using linear logic theorem proving', *Information Systems, Special Issue on the Semantic Web and Web Services*, Vol. 31, Nos. 4–5, pp.340–360.

Sewell, P., Wojciechowski, P.T. and Unyapoth, A. (2010) 'Nomadic pict: programming languages, communication infrastructure overlays, and semantics for mobile computation', *ACM Transactions on Programming Languages and Systems*, Vol. 32, No. 4, pp.1–94.

Wang, S., Zhu, X. and Yang, F. (2014) 'Efficient QoS management for QoS-aware web service composition', *International Journal of Web and Grid Services*, Vol. 10, No. 1, p.1.

Xiao, F.X., Li, Y., Huang, Z.Q. et al. (2012) 'Modeling and analyzing web services composition using timed probabilistic priced process algebra', *Chinese Journal of Computers*, Vol. 35, No. 5, pp.918–936.

Yun, B. (2013) 'Formal modeling of trust web service composition using pi-calculus', *Telkomnika Indonesian Journal of Electrical Engineering*, Vol. 11, No. 8, pp.4385–4392.

Zhang, Y.Y., Liu, J., Tang, Q. and Wu, Y. (2011) 'Modeling web services composition with timed pi calculus', *Information Technology Journal*, Vol. 10, No. 6, pp.1194–1200.