A Method of Improving QoS: Explorations of the Possibility of Function Combinations in Service Compositions
品質改良のアプローチ：機能のぶれを許すサービス合成

by

Ziyuan Wang
王子源

A Senior Thesis
卒業論文

Submitted to
the Department of Information Science
the Faculty of Science, the University of Tokyo
on February 9, 2016
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Shinichi Honiden　本位田真一
Professor of Information Science

ABSTRACT

There is a growing need for web service providers to develop customised and flexible web services as quick as they can. One way to satisfy this demand is to utilise service compositions, which provide a method of consolidating several services to a richer service. Because of the uncertainty in the feasibility of the functional and QoS requirements, it is not guaranteed that service compositions succeed with suitable solutions.

One way to satisfy the given requirements is to control the given two kinds of requirements. There have been studies on methods of optimising the QoS with fixed functional requirements. However, the search space in that case is limited, which would possibly not include service compositions with better QoS whose functions are slightly different from the required one. Therefore, I focus on the possibility of improving the QoS by compromising on functions or exploring the possibility of function combinations. This enables better advice of service compositions to users. In this article, I propose a method of efficiently searching different possibilities to support users to balance functional requirements and QoS requirements.

論文要旨

近年，ウェブサービスプロバイダーにとって，ユーザーの好みに合わせた，柔軟なウェブサービスを短時間で作る必要性が増してきている．そのために，よく使われている手法として，複数のサービスを一つにまとめてより高機能なサービスを作るサービス合成と呼ばれる手法がある．サービス合成において，入力として受け取る機能や品質に関する要求を必ず満たせるとは限らないという問題がある．

機能と品質の両方を達成するために既存研究で提案されている手法として，その片方を固定して保証し，もう片方を最適化するというものがある．しかし，この手法では，機能を固定するために，探索空間が限られているため，要求された機能に妥協を許した場合に品質が大きく高まるような合成を見逃す可能性がある．よって，本論文では，ユーザーが高品質の合成を見つけることがより容易になるよう，機能を妥協し，要求された機能と少しのぶれを許すことで探索空間を増やすことによる QoS の最適化の可能性について議論した．本論文では，ユーザーが機能要求と品質要求のバランスを取れるように効率的に様々な合成の可能性を探索する手法を提案した．

Acknowledgements

I appreciate Prof. Ishikawa's help in improving the structure of the paper.

Contents

Chapter 1

INTREODUCTION

Web service, service-based system, and service composition are significant.

1.1 Background

1.2 Motivation

1.3 Contributions

1.3.1 EXTENDED COMPOSITION PROBLEM

1.3.2 ALGORITHM FOR EXTENDED COMPOSITION PROBLEM

1.3.3 Evaluation

[1].

Chapter 2

Preliminary

This section defines some key terms of formal research of the service composition that I continue to use in this paper.

2.1 Service

A service S is a reusable system that provides functionalities which are documented in a service description. This description defines a 5-tuple *IOPEQ = (S.I, S.O, S.P, S.E, S.Q)* where *S.I, S.O* are abbreviated from the required inputs and outputs of the S, *S.P, S.E* are the preconditions and effects which denote the necessary condition of utilising S and the consequence after running S, and *S.Q* is the set of the QoS attributes of the S.

In this paper, *S.P, S.E* are ignored therefore a service S is modelled only with 3-tuple *IOQ = (S.I, S.O, S.Q)*.

2.2 QoS

QoS is abbreviated from quality-of-service, that is, quality apart from the functionality the service can provides, such as the price, the execution time and the reliability of the service. The S.Q of a service S is consist of a number of QoS attributes which is normalized between 0 and 1, with 0 being the worst and 1 being the best.

2.3 Service Compliance

If there exists an output $o \in S.O$ of a service S is compatible with an input of $i \in S'.I$ of another service S', we say there is a service link between S and S', written as S $\rightarrow$ S'. That is, the type of o is the same or a subtype of i.

In this paper, a service is constrained to has only one input and one output, therefore S $\rightarrow$ S' iff:

$$o = S.O \ , \ i = S'.I \ , \ o \in i$$

2.4 I/O Plug-in match

Two functionality similar services, in this paper that denotes they are in the same task, may have two relations, defined as follow.

An Input Plug-in match $\sqsubseteq$ is a S $\times$ S relation that holds iff:

$$\text{S} \sqsubseteq \text{S'} \Leftrightarrow S.I \subset S'.I$$

In this paper, S $\sqsubseteq$ S' is referred to as S' is input-stronger than S, and S is input-weaker than S'.

An Output Plug-in match $\subset$ is a S $\times$ S relation that holds iff:

$$S \subseteq S' \Leftrightarrow S.I \subset S'.I$$

In this paper, S $\subseteq$ S' is referred to as S' is output-stronger than S, and S is output-weaker than S'.

2.5 Workflow

A workflow is a sequence of two or more linked services. The functions of a workflow depend on function combinations that consist of the input parameter of the workflow's first service and the output parameter of the workflow's last service. A workflow template contains service tasks instead of actual services. A task is characterised as an abstract functionality which can be replaced by an actual service. There are generally two ways to assign services to a task, one is to compare the functions of the services with the functional requirements of the task. The other is to collect services depend on documents such as service description, which are usually distributed with the service by provider. Finally, each task is assigned with a set of services which meet the functional requirements of the task.

Selection algorithms receive a workflow template with fixed functional requirement and a dozens of services, and select for each task of one or more services which guarantee the obtained workflow's QoS are optimised, then return the obtained workflow as an output.

2.6 Functional requirements

To a workflow template W, where FT = the first task of W, IT = the last task of W, its functional requirements f can be:

$$f = (i, o), where\ i \subset W.FT.I, o \subset W.IT.O$$

In this paper, f = (a set of W.FT.services, a set of W.IT.services)

2.7 QoS optimization

First, calculate the QoS vector Q of the workflow, computed on the basis of the types of QoS attributes and the control flow of the workflow.

Second, calculated value σ, which is the components of the obtained QoS vector Q are aggregated into a single value σ by applying a weighted sum.

2.8 Service composition

service composition[1]: a system to solve composition problem
existing composition problem definition:
input: a workflow template, a number of services, function requirements
output: QoS best workflow and its QoS
existing algorithm of service composition: GA, ACO,...

Chapter 3

Related work

Chapter 4

Approach

I introduced the core study of this paper, definition of Extended Composition Problem in the next section.
In the second section, I summerized concrete algorithm solving the problem mentioned above, and its optimized algorithm. Initially, instead of full search, skyline algorithm (introduced in subsection) could reduce run times effectively, thus facilitating feasibility of the whole system evidently. Furthermore, naive algorithm (introduced in subsection) is capable of calculating several basic variations in variation degree respectively in response to customers' choices, on the basis of skyline algorithm. Those calculation results are compared to generate Scheme, thus recommending the most proper results to customers. Next, development algorithm (introduced in subsection) is a modified version of naive algorithm. In this algorithm, basic variations are divided into several groups by different input requirements for Scheme generation. Thus compared with naive algorithm, it could reduce calculation much effectively.

4.1 Extended Composition Problem

4.1.1 Concept of Variation

To a workflow template W, where FT is the first task of W, LT is the last task of W, i is a set of W.FT.services, o is a set of W.LT.services, W's functional requirements f is (i, o), I put forward the concept of Variation:

$$Variation = (i', o'), where\, i' \subset P(Strong(i), Weak(i), i), o' \subset P(Strong(o), Weak(o), o)$$

In addition, P represents powerset. In the next, I am going to introduce the definition of Strong and Weak.

To a service s in the FT(first task) of a workflow template W, Strong(s) is defined as below:

$$Strong(s) = \{s' \in FT, s \sqsubseteq s'\}$$

To a service s in the LT(last task) of a workflow template W, Strong(s) is defined as below:

$$Strong(s) = \{s' \in LT, s \subseteq s'\}$$

To a service s in the FT(first task) of a workflow template W, Weak(s) is defined as below:

$$Weak(s) = \{s' \in FT, s' \sqsubseteq s\}$$

To a service s in the LT(last task) of a workflow template W, Weak(s) is defined as below:

$$Weak(s) = \{s' \in LT, s' \subseteq s\}$$

Function elevation is expected unless QoS were not reduced.

To a set of services S $\subseteq$ FT(first task) of a workflow template W, Strong(S) is defined as below:

$$Strong(S) = \cup(Strong(s \in S))$$

To a set of services S $\subseteq$ LT(last task) of a workflow template W, Strong(S) is defined as below:

$$Strong(S) = \cup(Strong(s \in S))$$

To a set of services S $\subseteq$ FT(first task) of a workflow template W, Weak(S) is defined as below:

$$Weak(S) = \cup(Weak(s \in S))$$

To a set of services S $\subseteq$ LT(first task) of a workflow template W, Weak(S) is defined as below:

$$Weak(S) = \cup(Weak(s \in S))$$

To a functional requirement FQ = (i, o), where i = a set of W.FT.services, o = a set of W.LT.services

4.1.2 Three degrees of Variation

There are 8 kind of basic Variation:

$$FQ_{(s,s)}, FQ_{(s,f)}, FQ_{(s,w)}, FQ_{(f,s)}, FQ_{(f,w)}, FQ_{(w,s)}, FQ_{(w,f)}, FQ_{(w,w)}$$

There are 3 degrees of Variation:

$$FQ_{compatible} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)}$$

where compatible means functional compatible

$$FQ_{trade-off} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)} \cup FQ_{(w,s)} \cup FQ_{(s,w)}$$

where trade-off means functional trade-off

$$FQ_{compromise} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)} \cup FQ_{(w,s)} \cup FQ_{(s,w)} \cup FQ_{(f,w)} \cup FQ_{(w,f)} \cup FQ_{(w,w)}$$

where compromise means functional compromise

4.1.3 extended composition problem definition

4.2 Algorithm for Extended Composition Problem

:

4.2.1 Skyline

4.2.2 Naive Variation Algorithm

4.2.3 Developed Variation Algorithm

Chapter 5

Experiments

5.1 Implementation

5.2 Extended Composition Problem

Evaluation of Proposal 1
How QoS can be improved by allowing compromise in functional requirements?

Performance of three Variation degree in 828 valid cases

	Improved scheme numbers	average improvement degree(vs fixed functional requirements)
$FQ_{compatible}$	453	5.089129%
$FQ_{trade\text{-}off}$	485	8.827387%
$FQ_{compromise}$	566	11.80829%

5.3 Algorithm

Evaluation of Proposal 2
How fast can the proposed algorithm solve the extended problem?
Skyline vs Full search:
Fig. x depicts the relationship of step numbers and runtime between full search and skyline algorithm.

Comparison of the search space and time of Full search and Skyline

	step numbers	runtime(second)
Full search	2999026	16.729
Skyline	1974.1	0.5128
Full search/ Skyline	1519.1864647181	32.6228549141966

- with skyline vs. proposed algorithm with skyline

Comparison of the search space and time of different Variation

	step numbers	runtime(second)
Fixed functional requiremnet	2113	0.569703
Naive Variation algorithm $FQ_{compatible}$	6550	2.017894
Naive Variation algorithm $FQ_{trade\text{-}off}$	8837	2.642321
Naive Variation algorithm $FQ_{compromise}$	13062	3.992750
Developed Variation algorithm $FQ_{compatible}$	4100	0.992360
Developed Variation algorithm $FQ_{trade\text{-}off}$	5456	1.231265
Developed Variation algorithm $FQ_{compromise}$	5867	1.263835

Chapter 6

Discussion

6.1 Proposal

6.1.1 Extended Composition Problem

6.1.2 Algorithm for Extended Composition Problem

6.2 Future Work

6.2.1 QoS constraints

6.2.2 adaptive

6.2.3 change scability

Chapter 7

Conclusion

References

[1] E. Goto. The parametron, a digital computing element which utilizes parametric oscillation. *Proceedings of the IRE*, 47(8):1304–1316, 1959.