# Transforming Abstract QoS Requirements, Preferences, and Logic Constraints for Automatic Web Service Composition

Incheon Paik*, Haruhiko Takada*, and Michael N. Huhns**
** School of Computer Science and Engineering*
*University of Aizu*
*E-mail: paikic@u-aizu.ac.jp, takada@ebiz.u-aizu.ac.jp*
*** Dept. of Computer Science and Engineering*
*University of South Carolina*
*E-mail: huhns@sc.edu*

## Abstract

*The constraints revealed during a logical composition of services are often too abstract for automatic service composition. The abstract constraints have to be transformed to concrete attributes. This research investigates semi-automatic transformation of intermediate constraints to concrete constraints for automatic service composition. It considers simultaneously a stack of composition attributes for QoS, preferences, and logic constraints.*

**Keywords**
Automatic Web Service Composition, Composition Attributes, Constraint Transformation, Semantic Web

## 1. Introduction

The goal of automatic Web service composition is to create new value-added services from existing Web services, resulting in more capable and novel services for users.

Automatic service composition usually follows a two-stage procedure: logical composition followed by physical composition. The abstract constraints during logical composition (LC) supplied by users or generated internally need to be translated into concrete constraints that can be used by a physical composer (PC). The transformation must occur automatically.

In this paper, a framework for the semiautomatic transformation of intermediate composition attributes in the LC to concrete version is introduced. A schematic of the framework is shown in Figure 1.

## 2. Automatic Service Composition Procedure

As described above, the creation of a new service via Web service composition is a process involving logical composition, concrete candidate service extraction, and physical composition that considers the composition attributes to produce a concrete workflow (Figure 1).

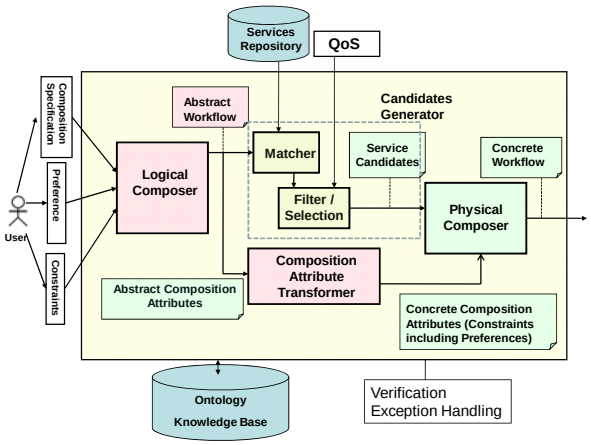


**Figure 1: A Framework for Automatic Service Composition**

### Attributes for Service Composition

The abstract workflow consists of solely the abstract process (or task) with its domain. There may be more than one abstract task, i.e., candidate services that satisfy QoS factors in that workflow. The PC selects a service considering all the constraints and preferences. The candidate services are to be selected by matched entries from a service registry.

To select concrete services from abstract services, the PC must consider user preferences and constraints, which are composition attributes. The composition attributes in the logical composition are abstract, so they do not match to real services that can be understood by a machine. They must therefore be transformed to a form that can be understood by the PC.

## 3. Composition Attribute Stack

The composition attributes come from users or domain-specific knowledge, and there are four levels of the attributes to consider. Level 0 can be handled by the LC and PC, so it is not included in the composition attributes that we are discussing. Level 1 describes QoS-related

information. These are used to generate candidate services for physical composition. Level 2 deals with domain-specific constraints or preferences. Level 3 deals with constraints and preferences beyond specific domains.

## 4. Transformation of Abstract Constraints

### 4.1 Abstract, Intermediate, and Concrete Constraints

Abstract constraints have informal concepts meaningful to humans. All terms are abstract and not already formalized into FOL.

Intermediate constraints consist of a relation, terms, and context information. They are generated by extracting abstract relations, terms, and context information from abstract terms (which may include context information) in natural language or compound terms at the upper level.

Concrete constraints have relations, terms as arguments of Web services, and invocation information for physical composition (in our implementation, CSP solving). All terms are bound to a real message for the operation of a Web service.

### 4.2 Ontologies to Support Transformation

The "AttributeDomain" ontology defines attributes for characteristics of terms that appear in constraints, such as for cost and time. The "ServiceDomain" ontology defines all the terminologies and relations of a service domain, such as the domains for travel planning or emergency situations. The "ContextInformation" ontology defines terms and operations for indicating a service that is related to the term that has an instance of the ontology. The instances of the ontologies are the terminal terms.

## 5. Transforming Architecture

We developed an architecture (Fig. 2) to transform an intermediate constraint into a concrete constraint. Sound transformation is carried out by a network of semantics and relations among classes in the ontologies for abstract/concrete constraint, abstract task, service candidate, and service/attribute domain. It includes an algorithm that uses the ontologies to find concrete terms from abstract terms.

### 5.1 Transformation Example

Our algorithm considers five cases according to the context information. As an illustration of a case in the algorithm, when a user wants to make a trip, the logical composer makes an abstract workflow for a trip sequence consisting of a train, an airplane, and a hotel; the tasks are

$task_n$ = {TrainFromAtoB, AirplaneFromBtoC, HotelC}

If a user wants the total cost to be less than $2,000, an abstract constraint is "*Total.Cost < 2000*".

As the "TermOperator" here means sum of candidates selected, it will be transformed to a concrete constraint

*(TrainSerivice.getCost("LocationA","LocationB") +*
*<AirlineB.getCost("BX101","Travel") ∨*
*AirlineB.getCost("BX102","Economy") ∨*
*AirlineC.getTicketFee("CZ103") > +*
*<HotelC1.getCost("Single") ∨ HotelC2.getPrice()> )*
*< 2000$*

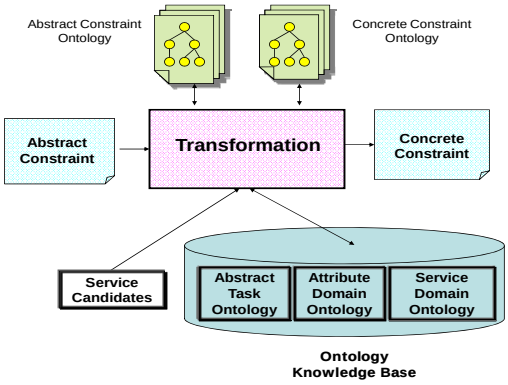


**Figure 2: A Transformation Architecture**

## 6. Conclusion and Future Work

We have developed an architecture of transforming abstract constraints into concrete constraints for automatic service composition. Our focus has been on abstract/concrete constraints together with ontology and an algorithm for transformation of the intermediate abstract constraints to concrete ones for automatic Web service composition. Future work will include translation from highly abstract constraints to intermediate abstract constraints and robust service orchestration.

## 7. References

[1] A.B. Hassine, S. Matsubara, and T. Ishida, A Constraint-based Approach to Horizontal Web Service Composition, *Proceedings of ISWC 2006*, pages 130-143, Athens, USA, 2006.

[2] V. Agarwal, and 6 others, A Service Creation Environment Based on End to End Composition of Web Services, *Proceedings of WWW 2005*, pages 128–137, Chiba, Japan, 2005.

[3] I. Paik, D. Maruyama, and M. Huhns, A Framework for Intelligent Web Services: Combined HTN and CSP Approach, *Proc. of IEEE ICWS*, pages 959-962, Chicago, 2006.