Composite Web Service Construction by Using a Logical Formalism

Pinar Senkul

Computer Engineering Department

Middle East Technical University

06530 Ankara, Turkey

karagoz@ceng.metu.edu.tr

Abstract

Today's Internet technology enables the customers to reach services through the web. However, some of the user demands may have a complex nature and can not be answered by a single web service. Composite services have to be constructed to fulfill complex service demands. Generally, these composite structures appear to be user specific. In addition to this, due to the highly dynamic nature of the web, it is not possible to determine the atomic services that will constitute the composition in advance. Therefore composite service model should include user's constraints and the composition should be performed dynamically and efficiently. The composition calls for the discovery of the services that fulfill the requirements of the user. In majority of the works on web service composition, the service discovery is based on service matching according to the requirements and constraints only on the atomic services. However, the requirements on the overall composition affect the selection of the services, as well. Another important point is to provide flexibility for the definition of the constraints. In this paper, we propose a system that can compose web services under the constraints on the overall composite service as well as requirements on the atomic services. We use a logic-based formalism, namely CCTR, for the definition of the composite web service and constraints, and the selection of the services. This framework provides a flexible constraint definition for the user by defining an acceptance level for constraints. The presented framework uses existing technologies for the discovery of services and for the service selection refinement and composition, CCTR scheduling mechanism is used.

1 Introduction

Web is growing to be a large collection of web services. There has been great efforts to define semantics and set standards for web services [35, 5, 6]. In the near future, it is

expected that most of the service demands from the users and the customers will be answered through the web. In order to answer complex demands, composite web services have to be constructed. A composite web service is a set of atomic services among which there is an execution and data flow. In addition to this, there may be a set of user-specified constraints on the atomic services and on the composition.

Building a composite service calls for the discovery of the atomic services in such a way that the constraints will be fulfilled. In most of the previous work, for the service selection and service matching, only the constraints on atomic services are considered. However, the constraints that are defined on the composition are also effective on the service selection. For example, for a celebration arrangement service that includes *send flower* and *dinner reservation* atomic services, a limitation on the total budget should eliminate composition of expensive services. On the other hand, if only one of the service is expensive, it may be included in the composite web service, as long as the total budget is within the given limits. Another example on constraint on composition is the *same constraint*, in which a user wants all of or a subset of the services of the composition to be fulfilled by the same service provider. Such a constraint may be necessary when working with the same service provider for several services have some benefits such as discount.

The user may not be quite sure for the limits while she/he defines the constraints. For example, the total budget and total quality constraints may form to a trade-off and the user may prefer to leave one for a stronger fulfillment in another. In relation with this issue, composition frameworks may do automatic or semi-automatic composition. In a semi-automatic composition, a set of plans is presented to the user and user selects the best one to be executed as the composite service.

In this paper, we present a system that can compose web services under the constraints on the overall composite services as well as the requirements on the individual services. We use a logic-based formalism, namely CCTR [26], for the definition of the composite service model and constraints

Figure 1. Architecture of the Framework

and the selection of the services. In this framework, the specification of cost constraints include *acceptance level* definition, which provides flexibility for the user to specify the limits. On the basis of the constraints and acceptance levels, a set of feasible plans are generated and ranked. This system may work as a automatic composition system where highest ranked plan is selected by default or it may work as semi-automatic system, where all plans are listed to the user and final decision is made by the user.

There has been extensive research on the service publish, discovery and message passing. This framework combines the existing technologies [32, 31, 36, 9, 16] for these steps with a new formalism for the composition. In this framework, service discovery is based on on UDDI [32] registries and the service selection is a two-level process. In the first level, service discovery is made through UDDI directories, on the basis of the required categories of the services. In this level, existing technologies for service matching and discovery can be incorporated into the system. In the second level, the matching service descriptions are translated to CCTR representations. CCTR scheduling mechanism is used to refine the service selection in order to guarantee that the services fulfill the constraints and acceptance level. As a result of this step, a set of plans is generated. The best plan is selected either by the system or by the user. Once the composite service is created, the definition can be translated into a service execution language such as WSFL [16] or BPEL4WS [9], for the execution.

The main contributions of this work can be listed as follows:

- The presented framework provides a uniform framework for the modeling and satisfaction of the constraints.
- It provides flexibility to the user for constraint specification

cation

- It can be used as an automatic or a semi-automatic composition system.

In the rest of the paper, the following is used as the running example.

Example 1.1 *For wedding anniversary celebration, a person wants to buy a flower bouquet, as a gift either buy a pair of earrings or buy a one-day weekend trip for his spouse and make dinner reservation. The constraints are budget ≤ 700 \$, duration ≤ 5 hours for the delivery of the flowers and the earrings (if it is chosen as the gift) and quality $\geq 3_star$ for dinner reservation.*

The organization of the paper is as follows. In Section 2, the architecture of the framework is presented. Section 3 explains the composite service modeling in CCTR. In Section 4, the refinement of selection and construction of the service is presented. Section 5 gives an overview of the related work and finally Section 6 concludes the paper with a summary of the work.

2 Architecture

The proposed system combines the existing technologies for service discovery with a logical framework for service selection refinement and web service composition. The architecture of the system is shown in Figure 1. In order to construct a composite service, the system follows the steps given below.

1. The general structure of the composite service and the constraints are defined by the user. The general structure includes the category templates for the web services. The web services that are compatible with the

template and fulfills the constraints are selected by the system and substituted. An abstract service includes the service type (category) information. The user may define further information and constraints on the service, such as inputs, service provider name, payment method. The kind of definition may vary with the service type.

2. Definitions provided by the user is converted to CCTR. Each service type instance is represented as a unique atomic formula in CCTR. The constraints definitions are translated to CCTR constraint predicates. The details of CCTR service definitions and constraint predicates are given in Section 3.2.
3. Definitions provided by the user is converted to service queries. Basically, for each service type that take part in the composite service definition, a query including a condition on the service type is produced. If the user provides additional information and constraints on the service, they are incorporated in the query as well.
4. Service matching is done according to the produced queries that have been described above. In our architecture, we use the UDDI [32] service directory as the service description repository. The category-based service selection is done through this directory. Additional conditions on the queries may refine the selection further. This step selects a set of candidate services for each abstract service that takes part in the composite service definition. These candidate service sets are stored in a local repository.
5. The selected services, together with their parameters are represented in CCTR. In CCTR terminology, the set of candidate services for a service template is called the *resources* for the service type. This step is explained in Section 3.2.
6. The set of candidate services fulfill the category and the other constraints defined on the abstract services. However, further refinement is necessary to select the set of services that fulfill the constraints on the composite service (e.g. same constraint, cost constraint). The CCTR scheduler refines the service selection on the basis of such constraints. The refinement affects the execution flow models that include Or/Xor branches. The plans generated by the CCTR scheduler include both the execution flow and service assignment. There may be more than one feasible plan and the generated plan set may include some plans that do not full comply the constraints, however, all the plans are within the acceptable limits provided by the user. The generated plan set is presented to the user in descending ranking.

7. The selected plan is converted to a composite web service. The resulting composite web service is constructed by substituting each service template with its corresponding service in the selected candidate service set. This definition can be translated to a web service execution language such as WSFL [16] or BPEL2WS [9].

In this paper, the emphasis is on the representation of the composite service model in CCTR and service selection refinement by CCTR scheduler. Therefore, in the following sections, we will present the steps 2, 5 and 6 in more detail.

3 Modelling the Composite Web Services

A composite web service model includes specification of the type of required services, the general flow of the services and the constraints on the composite service as well as on the individual services.

Definition 3.1 A composite web service *definition* includes

- *Abstract service definition: This definition is independent of the service instance. It basically includes the type/category information of the service. We assume that the categories are define on the basis of the standard service codes as in UNSPSC [33] or NAICS [21]. In addition, the input to the abstract service may be necessary for the discovery process. As an example, the for the earring buying service, the service type may be represented with 448310, which is the NAICS code for jewelry stores. As the input earring is defined as the input to the service definition.*
- *Flow definition: The flow definition includes the execution flow pattern. In this work, we limit the patterns to serial, concurrent and Or/Xor parallel structures. These structures are compatible with OWL-S composite process definition.*
- *Constraint definition: Constraint definitions include the constraints on the individual services and the constraints on the composite web service. The individual service constraints are defined on the parameters of the individual services such as payment method or service provider. Composite service constraints are cost constraints that are defined on one or more cost dimensions of the service, such as total budget or total duration. Cost constraint definitions are accompanied with one more parameter, called acceptance level. It provides relaxation on the constraints increasing the alternative for the user. The definition of acceptance level is given below.*

Definition 3.2 Given a cost constraint C in the form of $c < n$, acceptance level a for C is a numeric value such that $c < (n + a)$ becomes an acceptable constraint.

In this work, we limited the acceptance level definition to constraints that have an upper bound. Generally, relaxations on such type of constraints lead to more number of alternative plans.

In addition to the constraints defined by the user, there may be some implicit constraints as well. For instance, the compatibility of communication protocols of the web services that needs to pass parameters or compatibility of the parameter types are important constraints for the execution of the composite services. Such constraints can be defined as a default set of constraint definitions and added to the user's constraints automatically.

Previously, CCTR has been used to model and schedule workflows under resource allocation constraints. A composite web service share many common features with a workflow system. In a composite web service, individual services work in a coordinated and collaborative fashion as in the way tasks of a workflow execute. In this section, we first provide an overview of CCTR and then describe how CCTR is used as a formal framework for composite web service modeling.

3.1 CCTR

Concurrent Constraint Transaction Logic (CCTR) is an extension to Concurrent Transaction Logic (CTR) with the capability of workflow modeling and scheduling under resource allocation constraints. CTR extends first-order logic for programming, executing and reasoning state changing concurrent processes [8]. It has been successfully applied to modeling, reasoning about and scheduling workflows [11, 7]. CCTR exploits the workflow modeling, reasoning and scheduling capabilities of CTR and broadens these capabilities for the set of resource allocation constraints.

CCTR extends classical logic with new connectives and modalities. We present two of them which are important to modeling of workflows and e-services: $\otimes$, the *serial conjunction*, and $|$, the *parallel conjunction*. The simplest kind of a formula is an *atomic formula*, which has the usual form $p(t_1, \dots, t_n)$, where p is an n -ary predicate symbol taken from $\mathcal{P}$ or $\mathcal{C}$ and $t_1, \dots, t_n$ are terms. Complex formulas are constructed as follows: If ϕ and ψ are CCTR formulas, then so are the following expressions:

- $\phi \vee \psi, \phi \wedge \psi, \phi \otimes \psi, \phi | \psi, \neg \psi$
- $(\forall X)\phi$ and $(\exists X)\phi$, where X is a variable.

Intuitively, the formula $\phi \otimes \psi$ means that the subtransaction ϕ must execute first and the subtransaction ψ executes

next. The formula $\phi | \psi$ means that subtransactions ϕ and ψ execute concurrently.

The following formula models a composite web-service in which $service_1$ and $service_2$ run in order and $service_3$ is executed in parallel with two other two services.

$$((service_1 \otimes service_2) | service_3) \wedge c_1 \wedge c_2$$

In the above formula, c_1 and c_2 are constraint predicates that models the resource allocation constraints on this composite service. The conjunction operator denotes that these constraints must be true along the whole workflow.

In CCTR, the formulas are satisfied along partial schedules. A partial schedule is a set of database states which also captures the information as to whether the state changes take place in sequence or concurrently.

CCTR includes resource and resource assignment as the formal parts of the logic. A *resource* is an object with the attributes *token* and *cost* and a *resource assignment* is a partial mapping from partial schedules to sets of resources. Hence, resource assignment defines the set of resources needed along an execution.

In CCTR, the semantics of the constraint predicates is captured by the *constraint universe* parameter of the language. Constraint universe contains the domains and the relations that are used to define the semantics of constraint predicates. For each n -ary constraint predicate symbol $c \in \mathcal{C}$, there is a $(n+2)$ -ary relation c_D in the constraint universe. The first one of the new arguments is a partial schedule and the second one is a resource assignment. These extra arguments allow us to relate the behavior of the constraints to the execution of workflows.

The model theory of CCTR tells whether a given resource allocation along a partial schedule satisfies a CCTR formula. Therefore, informally, the following says that the CCTR formula α is true along execution ω under resource assignment ξ .

$$M, \omega, \xi \models \alpha$$

CCTR scheduler finds the executions and resource assignments that obeys CCTR model theory. The detailed information on CCTR can be found in [26].

3.2 Modeling a composite web service in CCTR

In order to model the composite service in CCTR, we have to represent each component of composite service definition in Definition 3.1 in CCTR. These are the representation of abstract services, representation of the flow and the representation of the constraints.

Abstract Service Modeling: In CCTR, each abstract service call is represented by an atomic formula. In a composite service, the same abstract service may take place more than once. However, each occurrence represents a sepa-

$$\begin{aligned}
& budget_{\leq_{CD}}(\omega, \xi, 700, 50) = budget_cost(\omega, \xi) \leq 700 + 50 \\
& budget_cost(\omega_1 \parallel \omega_2, \xi) = budget_cost(\omega_1, \xi) + budget_cost(\omega_2, \xi) \\
& budget_cost(\omega_1 \bullet \omega_2, \xi) = budget_cost(\omega_1, \xi) + budget_cost(\omega_2, \xi) \\
& budget_cost(\omega, \xi) = \xi(\omega).price \text{ (when } \omega \text{ is a path)} \\
& (\xi(\omega).price \text{ denotes the cost of resource assignment along simple executions, i.e, atomic services}) \\
\\
& duration_{\leq_{CD}}(\omega, \xi, 5, 0) = duration_cost(\omega, \xi) \leq 5 + 0 \\
& duration_cost(\omega_1 \parallel \omega_2, \xi) = \max(duration_cost(\omega_1, \xi), duration_cost(\omega_2, \xi)) \\
& duration_cost(\omega_1 \bullet \omega_2, \xi) = duration_cost(\omega_1, \xi) + duration_cost(\omega_2, \xi) \\
& duration_cost(\omega, \xi) = \xi(\omega).duration \text{ (when } \omega \text{ is a path)} \\
\\
& quality_{\geq_{CD}}(\omega, \xi, 3) = quality_cost(\omega, \xi) \geq 3 \\
& quality_cost(\omega_1 \parallel \omega_2, \xi) = quality_cost(\omega_1, \xi) + quality_cost(\omega_2, \xi) \\
& quality_cost(\omega_1 \bullet \omega_2, \xi) = quality_cost(\omega_1, \xi) + quality_cost(\omega_2, \xi) \\
& quality_cost(\omega, \xi) = \xi(\omega).quality \text{ (when } \omega \text{ is a path)}
\end{aligned}$$

Figure 2. Constraint Universe Definitions for Constraint Predicates

rate instance. For this reason, in CCTR, they are represented by different atomic formulas. For example, the service templates *flower-sending*, *buy-earrings-gift*, *buy-a-trip* and *dinner-reservation* of Example 1.1 are represented by the atoms *f*, *g*, *t* and *d*, respectively.

Flow Modeling: The operators of CCTR are used for modeling the control flow of the service. The sequential flow, concurrent flow and alternatives (Or branches) are represented by the operators $\otimes$, $|$ and $\vee$, respectively. We can represent the control flow in Example 1.1 as $(f \otimes (g \vee t)) \mid d$.

Constraints Modeling: The constraints on the composite service is represented as constraint predicates. The acceptance level is incorporated into the definitions as well.

Example 3.3 *The constraints of Example 1.1 are represented as CCTR constraint predicates as follows:*

- $budget_{\leq}(700, 50)$ denotes total budget should be less than \$700, with the acceptance level \$50.
- $duration_{\leq}(5, 0)$ denotes total duration should be less than 5hours with the acceptance level 0.
- $quality_{\geq}(3)$ denotes that the quality of the service, on which the constraint applied, should be equal to or above 3.

In CCTR, the semantics of the constraint predicates is defined in constraint universe. Constraint universe includes relations that are associated with constraint predicates.

Example 3.4 *The relations that are shown in Figure 3 define the semantics of the constraint predicates $budget_{\leq}$, $duration_{\leq}$ and $quality_{\geq}$ of Example 3.3. In order to accumulate the price cost of each simple execution, the relation*

$budget_{\leq_{CD}}$ makes use of a function named $budget_cost$. This function adds up the prices along concurrent and sequential executions. The definition of $duration_{\leq_{CD}}$ is also similar, except that the auxiliary function accumulates the duration costs differently. This time, the costs are added up for sequential executions, however, for concurrent executions, among the concurrent branches, maximum of the costs is selected.

It is possible to provide user templates for the definition of common types of constraints. In addition to this, complete description of a set of default constraints may be extracted from the process model and presented to the user.

4 Selection Refinement and Construction of Composite Web Services

As the result of service discovery step, a set of candidate services is formed for each template. However, at this point, it is still not clear whether they satisfy the constraints on the composite service. We need a selection refinement mechanism to guarantee that the service selection satisfies all of the given constraints. We use CCTR scheduler as the selection refinement module.

4.1 Service Discovery

The first step in the service selection is to discover a set of services for each service template. For this purpose, a category based search is applied on the service repository. In some cases, besides category type, it may be necessary to make selection on the basis of further constraints such as duration or payment method of the individual services. Such constraint are incorporated into the service queries.

Figure 3. Mapping from candidate service set to CCTR resource model for Example 1.1

CCTR scheduler can solve these constraints as well, however, it is beneficial to reduce the size of candidate set in earlier step. For such cases, the needed information has to be retrieved from the service and if it satisfies constraints that are defined on the service template, it is selected. In this step, it is also possible to adopt results of the research on the semantically enriched repositories [30, 12, 14, 15] for a more intelligent search. The details of service discovery is beyond the scope of this paper. As a result of this step, for each service template instance, a set of candidate services is obtained.

4.2 Selection Refinement

In CCTR, a resource is modeled as an object with one or more cost attributes. However, OWL-S based service descriptions are more complex and include many attributes. Each service in a candidate set is mapped to a resource model. In this mapping we keep only the relevant attributes of the services. Relevant attributes are the attributes of a service on which some constraint is defined. An example mapping for the candidate set generated for Example 1.1 is shown in Figure 3.

CCTR scheduler receives the resulting resource definitions, composite service model and constraints and produce an execution ordering and the service assignments to the templates that will constitute the composition. The scheduler is shown in Figure 4. The major steps of scheduling process is as follows:

- **Transformation:** CCTR scheduler includes CTR interpreter¹. In transformation step, CCTR formula is

¹CCTR interpreter is the implementation of CTR proof theory. Incorporating CTR interpreter in the scheduler provide reusability of the previously defined module

translated into a form that is compatible with CTR interpreter.

- **CTR Interpreter:** CTR interpreter produces possible execution orderings. This is crucial for flow structures that include Or/Xor branches. Selection of the execution branch among the alternatives may effect the satisfaction of the constraints. For this reason, the service filtering is done for each possible ordering that include different branch selection among alternative Or/Xor branches.
- **Constraint Solver:** CTR interpreter does not have the capability to solve the constraints. It accumulates the atomic constraint formulas and forward it to the constraint solver. The constraint solver produces a list of resource objects that includes a service for each service template in the execution ordering which is produced in the previous step.

Through backtracking, CCTR scheduler can produce all possible execution sequences and corresponding service assignments. By this way, we obtain all of the feasible plans.

Example 4.1 The original CCTR formula that models the composite service is as follows:

$$((f \otimes (g \vee t)) \mid (d \wedge \text{quality}_{\geq}(3))) \wedge \text{budget}_{\leq}(700, 50) \wedge \text{duration}_{\leq}(5, 0)$$

After the transformation, the following formula is generated:

$$(((f \otimes g) \mid (d \otimes \overline{\text{quality}_{\geq}(3, D)}))) \otimes \frac{\text{budget}_{\leq}(700, 50, ((F \bullet_p G) \parallel_p D))}{\text{duration}_{\leq}(5, 0, ((F \bullet_p G) \parallel_p D))} \vee (((f \otimes t) \mid (d \otimes \overline{\text{quality}_{\geq}(3, D)}))) \otimes \frac{\text{budget}_{\leq}(700, 50, ((F \bullet_p T) \parallel_p D))}{\text{duration}_{\leq}(5, 0, ((F \bullet_p T) \parallel_p D))} \otimes$$

Figure 4. CCTR Scheduler

$$\overline{duration}_{\leq}(5, 0, ((F \bullet_p T) \parallel_p D)))$$

The transformation eliminates $\wedge$'s and the formula becomes compatible with the CTR interpreter. The constraint predicates $\overline{budget}_{\leq}$, $\overline{duration}_{\leq}$ and $\overline{quality}_{\geq}$ are transformed to the classical predicates $\underline{budget}_{\leq}$, $\underline{duration}_{\leq}$ and $\underline{quality}_{\geq}$, respectively. For the constraint solving step, we need the semantics of the constraints. The relation definitions in the constraint universe is rewritten as a set of predicates so that the logic-based constraint solving engines can run on these rules. For example, the relation $\overline{budget}_{\leq_{CD}}(\omega, \xi, 700, 0)$ is mapped to the following definition of the predicate $\underline{budget}_{\leq}(700, 0, X)$ as follows:

$$\begin{aligned} \underline{budget}_{\leq}(700, 50, X) : - \\ & \quad \underline{budget_cost}(X, Cost), Cost \leq 700 + 50. \\ \underline{budget_cost}(X_1 \bullet_p X_2, Cost) : - \\ & \quad \underline{budget_cost}(X_1, Cost_1), \underline{budget_cost}(X_2, Cost_2), \\ & \quad Cost \text{ is } Cost_1 + Cost_2 \\ \underline{budget_cost}(X_1 \parallel_p X_2, Cost) : - \\ & \quad \underline{budget_cost}(X_1, Cost_1), \underline{budget_cost}(X_2, Cost_2), \\ & \quad Cost \text{ is } Cost_1 + Cost_2 \\ \underline{budget_cost}(X_{var}, Cost) : - \\ & \quad \underline{budget_cost_Of}(X, Cost), \text{ where } X_{var} \text{ is a variable.} \end{aligned}$$

The predicates $\underline{duration}_{\leq}$ and $\underline{quality}_{\geq}$ are defined in a similar way. Under these predicate definitions, the scheduler produces a set of feasible plans for the modeled composite web service. The set of feasible plans for our example is given in Figure 5. Notice that, due to the existence of or branch, the composition model $(f \otimes (g \vee t)) \mid d$ has two alternative execution orderings: $(f \otimes g) \mid d$ and $(f \otimes t) \mid d$. In Figure 5, the plans that include gift service have the flow structure $(f \otimes g) \mid d$, whereas the others have $(f \otimes t) \mid d$ as the flow structure.

4.3 Ranking the Plans

For most of the composite service descriptions, more than one plan is generated. These plans should be ranked according to some evaluation measure. In this work, we use the following evaluation function:

$$eval(p) = \sum_{i=1}^n (c_i - t_i) * w_i + \sum_{j=1}^m (t_j - c_j) * w_j$$

The evaluation measure consists of the sum of two calculations. The first part calculates the distance of the results from the threshold for constraints having an upper bound (e.g. $c < n$). The second one is for the distance calculation for constraints with lower bound (e.g. $c > m$). In this formula, n and m denote the number of constraints, having upper bound and lower bound, respectively. c_i (c_j) denotes the normalized *computed* value for constraint i (j), t_i (t_j) denotes the normalized *threshold (bound)* value and w_i (w_j) shows the weight of the constraint i (j). Weight values may be obtained statistically, on the basis of the previously preferred plans, or it may be provided by the user. (Default weight value is set as 1 for all weights). The weights are used for specifying priorities to the constraints. They reflect user's order of importance for the fulfillment of the constraints and facilitate the ranking. Note that, the constraints that require exact matching, such as "the payment method should be visa" are satisfied by all plans equally, therefore they do not need to participate in the ranking. The plans are ranked in the increasing order of their *eval* value. It is possible to choose the best one (the one with rank 1) and execute it automatically. However, the user's preferences may not match the weight values. Therefore, we present the ranked plans to the user for the final decision.

Another reason for preferring the user-decision is the existence of an acceptance level for the services. Due to the acceptance level definition, the list of feasible plans include the ones that do not fully comply with the constraints but still stay within the acceptance limits. For instance the total cost of last three plans of Figure 5 exceed the budget,

	f	t	g	d	quality of d	total budget	total dur.	
<i>plan</i> ₁	<i>fl</i> ₁	—	<i>gift</i> ₁	<i>res</i> ₁	3-star	550	4	
<i>plan</i> ₂	<i>fl</i> ₁	—	<i>gift</i> ₁	<i>res</i> ₂	4-star	650	4	
<i>plan</i> ₃	<i>fl</i> ₁	<i>trip</i> ₁	—	<i>res</i> ₁	3-star	650	2	
<i>plan</i> ₄	<i>fl</i> ₂	<i>trip</i> ₂	—	<i>res</i> ₁	3-star	700	2	
<i>plan</i> ₅	<i>fl</i> ₁	—	<i>gift</i> ₁	<i>res</i> ₃	5-star	750	2	*
<i>plan</i> ₆	<i>fl</i> ₂	<i>trip</i> ₁	—	<i>res</i> ₂	4-star	750	2	*
<i>plan</i> ₇	<i>fl</i> ₁	<i>trip</i> ₂	—	<i>res</i> ₁	3-star	750	2	*

Figure 5. Plans for Example 1.1

however, *plan*₅ may be preferable, since it includes a 5-star restaurant with a slight increase in the budget. In the plan ranking, the plans that do not fully satisfy the constraints are marked for user's informations. (In Figure 5, such plans are marked with *). These plans are detected during *evaluation* calculation. Notice that, if a plan fails to satisfy constraint *i* (with upper bound), then $(c_i - t_i > 0)$.

Once the plan selection is completed, the composite service is constructed by substituting the service with the corresponding abstract model. The resulting composite service representation may be translated to some service execution language such as BPEL4WS.

5 Related Work

Web services have been a rich and challenging research area that. There are many dimensions to be studied. [10, 2, 27] present an overview of the web services and current technology. There has been standards proposed for service publish, registry and discovery, such as [32, 31, 36]. In this work, we include some of these standards and tools, especially for service discovery process.

The semantic description of the web services plays an important role in the machine-understandability which leads to the automation of the web service discovery, composition and execution. There are several languages for defining web service semantics such as OWL-S [17] and WSMO [37]. In [25] the relation between OWL-S and CCTR has been discussed. CCTR and OWL-S have complementary roles and functionalities for composite web services, where OWL-S describes semantics of the individual services and CCTR models the constraint, formal structure of the process and does scheduling under the constraints. In rule based language of WSMO, name WSDL, provides a declarative as in CCTR. However, the process modeling and the rules and constraints expressible on the process model is not presented in detail. WSDL-S [1] is another framework for web service semantic description. However, the main difference of WSDL-S from other semantic description languages is that it allows the portability of semantic description of web service to the service model. The WSDL

description is annotated with the semantic description in the language of user's choice. As a result, all such ontology languages are orthogonal to CCTR and can be used together with CCTR for concrete modeling.

There has been several related works on the composition of web services. [3] and [24] present detailed surveys on the web service composition approaches. In [34], a comparative study of the web service composition languages is given. Several works use different approaches and architectures for web service composition [22, 4, 20, 23, 28, 29, 13]. Among them, [22, 23] and [38] is similar to this work in the sense that they emphasize the constraints and constraint satisfaction methods. METEOR-S [22, 23] presents a detailed web service composition architecture. Constraint definitions on the composite service are used for optimal plan generation. Another work, [38], uses linear programming for solving constraints on composite service. In both of these approaches, separate tools are used for process modeling, constraint definition and satisfaction. The major difference in this work is the use of a logic-based formalism in which both process model and constraints can be specified and checked. It provides a uniform framework for modeling and service refinement. The semantics of the constraints on the services and on the composition can be defined by the user. The logic-based scheduler can detect the effect of constraints on the flow structure and the generated plans include both flow and service assignment information.

In [19, 18], as in this work, a logical formalization is used for the composite service modeling. This formalization declarative defines the relations between the atomic services and can express and reason about constraints on the atomic services. In our work, the set of expressible constraints is larger, including constraints on the overall composite structure. In addition to this, the scheduler includes a constraint solver for checking the satisfiability of constraints efficiently and for filtering the candidate services for the constraint satisfaction.

In [28, 29], some methods for semi-automatic composition are presented. In these works, an architecture through which user can define constraints on the individual services dynamically, is proposed. In this work, in contrast

to [28, 29], user's contribution to the system is at the point of plan selection. It is a semi-automatic system in the sense that user contributes to the plan selection. The system can be tuned to work as an automatic system by the help of plan ranking, as well. It may be an interesting research direction to combine these two different semi-automatic composition approaches.

6 Conclusion

The Internet is growing to be a big service repository and most of the the business and non-business services are expected to be served through the web in the future. As a result of this trend, web service composition becomes inevitable in order to answer complex requirements of the user. Composition of the services include many challenging tasks such as modeling, service discovery, service compatibility, service selection, plan generation and execution. In this work, the emphasis is in constraint modeling, service selection refinement under the the constraints and plan generation.

In this paper, we present a logic-based composite web service modeling and construction framework. It provides a formal and unified framework for representation and checking all constraints defined on the composite web service. We use the logic-based language CCTR for this purpose. CCTR's language primitives allow the modeling of the flow of the composite model and constraints on the model. CCTR scheduler produces feasible plans fulfilling the constraints. Another contribution of the work is definition of *allowance level* for constraints. This provides a flexibility for the user to define constraints and to select the best composition plan.

In the implementation of the architecture, we are at the phase of selection refinement and composition modules development. We plan to extend this work with an ontology-based service discovery module, execution engine and monitoring of the composite service.

References

- [1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. Schmidt, A. Sheth, and K. Verma. Web service semantics - wsdl-s. A joint UGA-IBM Technical Note, version 1.0, April 2005.
- [2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture, and Applications*. Springer Verlag, June 2003.
- [3] B. Benatallah, M. Dumas, M.-C. Fauvet, and F. A. Rabhi. Towards patterns of web service composition. In *Patterns and Skeletons for Parallel and Distributed Computing*, pages 265–296. Springer-Verlag, 2003.
- [4] B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv environment for web service composition. *IEEE Internet Computing*, 7(1):40–48, 2003.
- [5] T. Berners-Lee. Semantic web road map. <http://www.w3.org/DesignIssues/Semantic.html>, September 1998.
- [6] T. Berners-Lee. The semantic toolbox: Building semantics on top of XML-RDF. <http://www.w3.org/DesignIssues/Toolbox.html>, May 1999.
- [7] A. Bonner. Workflow, transactions, and datalog. In *ACM Symposium on Principles of Database Systems*, Philadelphia, PA, May/June 1999.
- [8] A. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint Int'l Conference and Symposium on Logic Programming*, pages 142–156, Bonn, Germany, September 1996. MIT Press.
- [9] Business Process Execution Language for Web Services (BPEL4WS). <http://xml.coverpages.org/bpel4ws.html>.
- [10] F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In *OOPSLA 2001 Workshop on Object-Oriented Web Services*, 2001.
- [11] H. Davulcu, M. Kifer, C. Ramakrishnan, and I. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*, pages 25–33, Seattle, Washington, June 1998.
- [12] A. Dogac, I. Cingil, G. B. Laleci, and Y. Kabak. Improving the functionality of uddi registries through web service semantics. In *3rd VLDB Workshop on Technologies for E-Services (TES-02)*, Hong Kong, China, August 2002.
- [13] A. Dogac, Y. Kabak, and G. Laleci. A semantic-based web service composition facility for ebxml registries. In *9th International Conference of Concurrent Enterprising*, Espoo, Finland, June 2003.
- [14] A. Dogac, Y. Kabak, and G. Laleci. Enriching ebxml registries with owl ontologies for efficient service discovery. In *14th International Workshop on Research Issues on Data Engineering (RIDE)*, pages 69–76, 2004.
- [15] A. Dogac, Y. Kabak, G. B. Laleci, C. Mattocks, F. Najmi, and J. Pollock. Enhancing ebxml registries to make them owl aware. *Distributed and Parallel Databases Journal*, 18(1):9–36, July 2005.
- [16] F. Leymann. Web Services Flow Language (WSFL), Version 1.0. <http://www306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, 2001.
- [17] D. Martin and et. al. OWL-S: Semantic Markup for Web Services. <http://www.daml.org/services/owl-s/1.1/overview/>, 2004.
- [18] S. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *In Proc. of the 8th Int. Conf. on Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, April 2002.
- [19] S. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, March/April 2001.
- [20] B. Medjadedh, A. Bouguettaya, and A. K. Elmagarmid. Composing web services on the semantic web. *VLDB Journal*, 12(4), November 2003.
- [21] North American Industry Classification System (NAICS). <http://www.census.gov/epcd/www/naics.html>, 2002.
- [22] K. V. P. Rajasekaran, J. A. Miller and A. Sheth. Enhancing web services description and discovery to facilitate composition. In *First International Workshop on Semantic Web*

Services and Web Process Composition (SWSWPC), pages 34–47, San Diego, CA, USA, July 2004.

- [23] J. A. M. R. Aggarwal, K. Verma and W. Milnor. Constraint driven web service composition in meteor-s. In *Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 2004)*, pages 23–30, Shanghai, China, 2004.
- [24] J. Rao and X. Su. A survey of automated web service composition methods. In *In Proc. of First International Workshop of Semantic Web Services and Web Process Composition*, San Diego, California, July 2004.
- [25] P. Senkul. Modeling composite web services by using a logic-based language. In *Second Italian Workshop on Semantic Web Applications and Perspectives*, Trento, Italy, December 2005.
- [26] P. Senkul, M. Kifer, and I. H. Toroslu. A logical framework for scheduling workflows under resource allocation constraints. In *Int'l Conference on Very Large Data Bases*, Hong Kong, China, 2002.
- [27] A. Sheth. From semantic search and integration to analytics. In *Semantic Interoperability and Integration*, 2005.
- [28] E. Sirin, J. Hendler, and P. B. Semi-automatic composition of web services using semantic descriptions. In *In Proc. of Web Services: Modeling, Architecture and Infrastructure Workshop in conjunction with ICEIS 2002*, 2002.
- [29] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.
- [30] K. Sivashanmugam, K. Verma, and A. P. Sheth. Discovery of web services in a federated registry environment. In *IEEE International Conference on Web Services (ICWS)*, pages 270–278, 2004.
- [31] Simple Object Access Protocol (SOAP). <http://www.w3c.org/TR/SOAP/>, 2003.
- [32] Universal Description, Discovery and Integration (UDDI). <http://www.uddi.org>, 2003.
- [33] United Nations Standard Products and Services Code (UNSPSC). <http://www.unspsc.org/>, 2003.
- [34] W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Web service composition languages: Old wine in new bottles? In *EUROMICRO*, pages 298–307, 2003.
- [35] World Wide Web Consortium (W3C). <http://www.w3c.org>.
- [36] Web Service Description Language. <http://www.w3c.org/TR/wsdl/>, 2003.
- [37] Web Service Modeling Ontology (WSMO). <http://www.wsmo.org/>.
- [38] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web service composition. In *Twelfth International World Wide Web Conference (WWW)*, 2003.