Modeling Composite Web Services by Using a Logic-based Language

Pinar Senkul

Middle East Technical University
Computer Engineering Department
06531 Ankara, Turkey
karagoz@ceng.metu.edu.tr

Abstract. In order to answer the complex service requirements of the user, composite web services have to be constructed correctly and effectively. Various approaches and formalism have been used for web service composition and integration. The semantic modeling of composite services is necessary for automatic discovery, integration and execution. For this purpose, ontology languages and ontologies have been defined. OWL-S is a OWL-based ontology of services, in which composite processes can be modeled. For reasoning and verification on the composite services, logic-based formalisms have an important role. Concurrent Constraint Transaction Logic is a formalism that provides means for modeling, verification and scheduling of composite web services. In this work, we describe how OWL-S and CCTR can be used together for modeling a complex service and constraints, and make reasoning and verification on this model under the given set of constraints.

1 Introduction

A composite web service is a web process that consists of a set of atomic web services that execute in a collaborative way. Composite services are necessary when there is no single service that will answer a complex service requirement. Travel arrangements, on-line business processes and online insurance services are some typical examples. A user may request a complex service that will include various different atomic services. Each service may be provided by a set of providers. Therefore the possible combinations are numerous.

For automatic discovery, integration and execution of the web services, semantic modeling of the services is necessary. For this purpose, ontology languages and ontologies have been defined. OWL-S is a OWL-based ontology of services, including core set of mark-up language constructs for describing the properties and capabilities of the Web services in a machine-understandable form [11]. The ontology has three main parts: the service profile, the process model, and the grounding. The process definitions allows the composite service modeling. It supports the several control constructs for the composite service, including sequence, split and join, if-then-else and iterate. By this way OWL-S provides a semantic description for a composite model. This description may be used for composite service query and matching as well as invocation.

The composite service requirement is generally defined by a process model of the service and some constraints on this model. The process model defines the control flow among the individual services. The set of constraints that may be defined on the process model is quiet rich. On the basis of the experiences on workflow model, we can group the constraints as follows:

- Constraints on the ordering and existence of the atomic services
- Constraints on the service providers
- Constraints on the properties of the composite service

Given a composite model or a concrete composite service, it is necessary to verify its correctness under a given set of constraints. There are several formalisms that provide an environment for modeling and verification of composite services, such as [12]. In this paper, we present a logic-based formalism, called Current Constraint Transaction Logic (CCTR) and elaborate on how to use this formalism for modeling and verification of composite web services, together with the semantic model given in OWL-S.

Concurrent Constraint Transaction Logic (CCTR) is an extension to Concurrent Transaction Logic (CTR) [7] with the capability of modeling complex processes and scheduling under resource allocation constraints. CTR has been successfully applied to modeling, reasoning about and scheduling workflows [9, 6]. In CCTR, these capabilities of CTR have been extended for the set of cost constraints and resource allocation constraints and it has been used for modeling and reasoning on workflows under a rich set of constraints including ordering constraints, resource allocation and cost constraints [19].

CCTR includes constructs for the formal modeling of resources, allocation of resources along execution and constraints on the allocation of the resources and cost of the resource allocation as well as modeling a complex process. This formalism provides reasoning about the correctness of a complex process under a given set of constraints. We consider atomic web service providers as the resources required for a composite web service, and use CCTR to model and reason about the properties of composite web services and constraints defined on them.

CCTR provides

- specification of the control flow of a composite process through its serial and concurrent conjunction operators
- specification of temporal constraints on the composite model
- specification of resource constraints
- specification of cost constraints
- reasoning on the correctness of the model under the given set of constraints.

The semantic model provided by OWL-S and its advantages can be combined with the formal modeling and verification functionality of CCTR. The control constructs of OWL-S can be mapped to CCTR for reasoning and verification

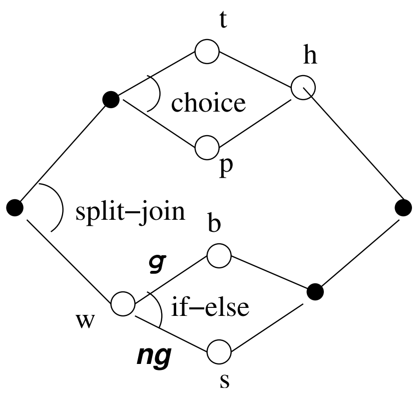

Fig. 1. Control flow of the Example 1

purpose and profile and grounding of the ontology and CCTR features can be used in complementary way. In this work, we present the functionalities of CCTR and elaborate on how these functionalities can be combined with the OWL-S semantic model.

In the rest of the paper, we will use the following as the running example.

Example 1. Assume that a person is planning a trip. First, she needs plane or train seat reservations and then hotel reservation. Meanwhile, she wants to check the weather forecast for the given time interval and make either skiing or balloon tour arrangements according to the weather forecast. She wants to do balloon tour reservation (if it will be chosen) after the hotel arrangement so that timings should match. In addition to this, there are limitations on the total budget and duration (for example traveling with plane may be better for budget considerations but takes longer time). The ordering of the services is graphically shown in 1. In this figure, the empty circles represent the services and black nodes denote synchronization points. The service are represented by the initials of the service type names. g and ng denotes the weather conditions *good* and *not good* that are obtained from the service w. For the control constructs such as concurrent execution or selection, we used the OWL-S construct names.

This paper is organized as follows: In Section 2 an overview of OWL-S is given. Section 3 describes CCTR and how it was used for workflow modeling. Section 4 elaborates on the relation between OWL-S and CCTR composite process models. Section 5 presents how to use CCTR for web service model verification. Section 6 presents the related work. Section 7 concludes the paper with a summary.

2 Overview of OWL-S

OWL-S [11] is a OWL-based [3] ontology of services, including a core set of mark-up language constructs for describing the properties and capabilities of the

```
⟨ process: CompositeProcess rdf:ID = "CompositeProc"⟩
        .....
⟨ process: ComposedOf ⟩
    ⟨ process: Split-Join rdf:ID = "SplitJoinCons"⟩
        ⟨ process: Sequence rdf:ID = "SeqCon" ⟩
            ⟨ process: Choice rdf:ID = "ChoiceCon" ⟩
                ⟨ process: Perform rdf:ID = "t" ⟩ .... ⟨/process: Perform ⟩
                ⟨ process: Perform rdf:ID = "p" ⟩ .... ⟨/process: Perform ⟩
            ⟨/process: Choice ⟩
            ⟨ process: Perform rdf:ID = "h" ⟩ .... ⟨/process: Perform ⟩
            ⟨/process: Choice ⟩
        ⟨/process: Sequence ⟩
        ⟨ process: Sequence rdf:ID = "SeqCon2" ⟩
            ⟨ process: Perform rdf:ID = "w" ⟩ .... ⟨/process: Perform ⟩
            ⟨ process: If-Then-Else rdf = "IfCons" ⟩
                ⟨ If-Then-Else: ifCondition rdf:ID = "g" ⟩ ... ⟨/If-Then-Else:... ⟩
                ⟨ If-Then-Else: then rdf:ID = "ThenPart" ⟩ ....
                    ⟨ process: Perform rdf:ID = "b" ⟩ .... ⟨/process: Perform ⟩
                ⟨/If-Then-Else: then ⟩
                ⟨ If-Then-Else: else rdf:ID = "ElsePart" ⟩ ....
                    ⟨ process: Perform rdf:ID = "s" ⟩ .... ⟨/process: Perform ⟩
                ⟨/If-Then-Else: else ⟩
            ⟨/process: If-Then-Else ⟩
        ⟨/process: Sequence ⟩
    ⟨/process: Split-Join ⟩
⟨ process: ComposedOf ⟩
        .....
⟨/process: CompositeProcess ⟩
```

Fig. 2. OWL-S Process Description for Example 1

Web services in a machine-understandable form. This machine-understandable form provides the semantic model that is necessary for the automatic discovery, integration and invocation of the services. The ontology has three parts: the *service profile*, the *process model* and the *grounding*. The service profile tells the functionality of the service. It includes the description of what service does, the properties and quality information. The service profile is the part that gives information about the service to a service seeker. The grounding specifies how to access the service by providing information on protocols, message formats and other accessing details.

The service model is the part that provides the information on the structure and semantics of the service. In this part, the inputs, outputs, preconditions and effects. OWL-S supports specification of both *atomic* and *composite processes*. The third type, *simple process* provides an abstraction on atomic process and can be substituted with a concrete atomic process description.

The composite process includes a set of control constructs to specify the ordering and existence relationships between the subprocesses. These constructs are sequence, split, split-join, any-order, choice, if-then-else, iteration, repeat-while and repeat-until. The composition of the processes are defined in a recursive way such that each subprocess may be further decomposed into parts that are combined with these control constructs until possibly an atomic model is reached. A simplified OWL-S modeling of Example 1 is given in Figure 2.

The process also model defines data flow and input-output bindings between the subparts. However, in the scope of this work, the focus is on the composite model built by using the control constructs. In the rest of the paper, we discuss how the these constructs can be mapped to a logical formalism for verification and further reasoning on the model.

3 Concurrent Constraint Transaction Logic (CCTR)

3.1 CCTR Overview

Concurrent Constraint Transaction Logic (CCTR) is an extension to Concurrent Transaction Logic (CTR) with the capability of modeling entities that will execute the tasks (i.e., resources) as a part of formalization, constraint specification on the resources and reasoning and scheduling under these constraints. One of the basic application areas is the workflow modeling and scheduling under resource allocation constraints [19]. CTR, itself, is an extension to first-order logic for programming, executing and reasoning state changing concurrent processes [7]. It has been successfully applied to modeling, reasoning about and scheduling workflows [9, 6].

CCTR extends classical logic with new connectives and modalities. We present three of them which important to modeling of workflows and composite web services:

- $\otimes$, the *serial conjunction*: it denotes the serial execution of subprocesses.
- $|$, the *parallel conjunction*: it denotes the concurrent execution of subprocesses in an interleaved way.
- $\odot$, the *atomic execution*: it denotes the atomic execution of a subprocess without interleaving with other executions.

The simplest kind of a formula is an *atomic formula*, which has the usual form $p(t_1, ..., t_n)$, where p is an n-ary predicate symbol taken from $\mathcal{P}$ or $\mathcal{C}$ and t_1, ..., t_n are terms. $\mathcal{C}$ is a set of special predicates called *constraint predicates* that are used for resource allocation constraint specification. Complex formulas are constructed by a set of recursive definitions. If ϕ and ψ are CCTR formulas, then so are the following expressions:

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

In CCTR, the correctness of the formulas are found against *partial schedules* (as opposed to states in classical logic). A partial schedule is a structure that represents the state changes through serial and concurrent executions. In CCTR, *state* is a set of objects and a partial schedule is a more structured form of *path* and *m-paths.* A path is a sequence of database states that denotes serial executions and m-path is a sequence of paths to denote interleaved execution of concurrent processes. In partial schedule, concurrent and serial subparts of the executions are explicitly shown in order to reason about correctness of the resource allocation constraints along the execution. $\langle D_0 D_1 D_2 \rangle$, $\langle D_0 D_1 D_2 D_3 \rangle$ and $\langle D_0 D_1 \rangle \parallel_p \langle D_1 D_2 \rangle$ are examples for path, m-path and partial schedule, respectively. In a partial schedule, $\parallel_p$ combines the m-paths that belong to interleaved executions and $\bullet_p$ combines the m-paths that belong to the serial executions.

In CCTR, resource and resource assignment are 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.

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 has a scheduler that finds the executions and resource assignments (together they can be considered as a plan) that conforms to the model theory. The detailed information on CCTR can be found in [19].

3.2 Modeling Workflows and Composite Web Services in CCTR

The new connectives of CCTR supports the modeling of serial and concurrent combinations of the state changes and, by this way, provides a formal modeling environment for composite processes such as workflows and composite web services. In addition to modeling of the control flow through these connectives, the formal representation of resource and resource constraint specifications enables the users to reason about the required conditions on the resources of the composite processes.

Consider a workflow in which a $task_1$ and $task_2$ run in order and $task_3$ is executed concurrently with two other two services. The following formula captures the given ordering relations:

$$(task_1 \otimes task_2) \mid task_3$$

Assume that on this workflow we define a set of resources to execute the tasks and enforce the following constraints: *the concurrent tasks should be executed by different resources*, *the total duration should not exceed* n and $task_3$ *must be* *executed before* $task_2$. The following formula extend the above flow definition with the constraints.

$((task_1 \otimes task_2) \mid task_3) \wedge c_1 \wedge c_2 \wedge c_3$

In the above formula, c_1, c_2 and c_3 are constraint predicates that models the resource allocation constraints and temporal constraint on the workflow that have been stated above. The conjunction operator denotes that these constraints must be true along the execution of the whole workflow.

CTR proof theory can not handle the formulas that include conjunction $\wedge$. However, in [9], an algorithm is presented to test the compatibility of an ordering constraint with a given formula. Similarly, CCTR does not have a proof theory that can do reasoning on conjuncted formulas. However, it has a scheduler that transforms the conjuncted formulas into non-conjuncted forms and reason about the correctness of resource constraints on the transformed formula. If the initial formula and the constraints are compatible, scheduler produces execution ordering and resource assignments for the produced ordering. Verification and reasoning on temporal and resource constraints are discussed in more detail in Section 5 in composite web service context.

4 From OWL-S to CCTR

OWL-S composite process model supports the representation of the following control/flow constructs: sequence, split, split and join, any order, choice, if-then-else, iterate, repeat-while and repeat-until.

In this section, we describe how each of these constructs can be represented in CCTR. In OWL-S modeling abstract level, simplest service model s_{owl}, represents an abstract service with the ontology information of the required service. In CCTR, this corresponds an atomic formula, s_{cctr}. Atomic formulas may be updates (world-altering tasks) or just queries (non-world-altering tasks. We represent each abstract service as an update. However, this update is only limited with insertion of significant event informations into the logs (such as *start, commit*).

4.1 Sequence

Sequence denotes a list of constructs to be done in order. Consider the following generalized sequence construct representation in OWL-S:

```
⟨ process: Sequence rdf:ID = "SeqCons"⟩
    ⟨ process: Perform rdf = "S1" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨ process: Perform rdf = "S2" ⟩ ..... ⟨ / process: Perform ⟩
      .....
    ⟨ process: Perform rdf = "Sn" ⟩ ..... ⟨ / process: Perform ⟩
      .....
⟨ /process: Sequence ⟩
```

Sequential composition is represented with the *serial conjunction* ($\otimes$) operator of CCTR. CCTR representation of *SeqCons* is as follows:

$S1 \otimes S2 \otimes ... \otimes Sn$

4.2 Split-Join

Split and join construct consists of a bag of process components to be executed concurrently. *Split-Join* completes when all of its subcomponents complete. Consider the following generalized split-join construct representation in OWL-S:

```
⟨ process: Split-Join rdf:ID = "SplitJoinCons"⟩
    ⟨ process: Perform rdf = "S1" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨ process: Perform rdf = "S2" ⟩ ..... ⟨ / process: Perform ⟩
      .....
    ⟨ process: Perform rdf = "Sn" ⟩ ..... ⟨ / process: Perform ⟩
      .....
⟨ /process: Split-Join ⟩
```

Split-join composition is represented with the *concurrent conjunction* ($|$) operator of CCTR. CCTR representation of *SplitJoinCons* is as follows:

$S1 \mid S2 \mid ... \mid Sn$

In CCTR, there is no operator that correspond to *Split* construct, however partial synchronizations can be modeled by split-join and order constraints.

4.3 Any-Order

Any-order denotes a set of constructs to be executed in some unspecified order. The sub processes can not execute concurrently or in an interleaved fashion. *Any-Order* completes when all of its subcomponents complete. Consider the following generalized any-order construct representation in OWL-S:

```
⟨ process: Any-Order rdf:ID = "AnyOrderCons"⟩
    ⟨ process: Perform rdf = "S1" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨ process: Perform rdf = "S2" ⟩ ..... ⟨ / process: Perform ⟩
      .....
    ⟨ process: Perform rdf = "Sn" ⟩ ..... ⟨ / process: Perform ⟩
      .....
⟨ /process: Any-Order ⟩
```

The semantics of concurrent conjunction ($|$) operator of CCTR is based on interleaving of concurrent subformulas. Another CCTR operator, atomic action ($\odot$), specifies the cases where the execution should be *atomic*, i.e., it *should not* *be interleaved* with other executions. By using these operators, CCTR representation of *SeqCons* is as follows:

$\odot(S1) \mid \odot(S2) \mid ... \mid \odot(Sn)$

4.4 Choice

Choice construct represents the execution of a single construct from the bag of subprocesses. Any of the subprocesses can be chosen for execution. Consider the following generalized choice construct representation in OWL-S:

```
⟨ process: Choice rdf:ID = "ChoiceCons"⟩
    ⟨ process: Perform rdf = "S1" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨ process: Perform rdf = "S2" ⟩ ..... ⟨ / process: Perform ⟩
      .....
    ⟨ process: Perform rdf = "Sn" ⟩ ..... ⟨ / process: Perform ⟩
      .....
⟨ /process: Choice ⟩
```

This construct can be modeled with *or* ($\vee$) operator of CCTR. *ChoiceCons* construct can be represented in CCTR as follows:

$S1 \vee S2 \vee ... \vee Sn$

4.5 If-Then-Else

If-Then-Else construct models the conventional "if cond then execute task a else execute task b" statement of programming languages, meaning that if the given condition is true execute *task a*, otherwise execute *task b*. Consider the following generalized if-then-else construct representation in OWL-S:

```
⟨ process: If-Then-Else rdf:ID = "IfCons"⟩
    ⟨ If-Then-Else: ifCond rdf = "Cond" ⟩ ..... ⟨ / If-Then-Else: ifCond ⟩
    ⟨ If-Then-Else: then rdf:ID = "ThenPart" ⟩ ....
        ⟨ process: Perform rdf = "S1" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨/If-Then-Else: then ⟩
    ⟨ If-Then-Else: else ⟩ ....
        ⟨ process: Perform rdf = "S2" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨/If-Then-Else: else ⟩
      .....
⟨ /process: If-Then-Else ⟩
```

This construct can be modeled by using an atomic query (i.e non-world-altering) task and *or* ($\vee$) operator of CCTR. *IfCons* construct can be represented in CCTR as follows:

$$(Cond \otimes S1) \vee ((\neg Cond) \otimes S2)$$

4.6 Iterate

Iterate construct represents the iteration of a process at an abstract level without considering the number of iterations or stopping condition for iterations. Iteration is generally instantiated with the more concrete subclasses *Repeat-While* and *Repeat-Until*

Repeat-While *Repeat-While* construct models the iteration that continues as long as the given condition is satisfied. Consider the following generalized repeat-while construct representation in OWL-S:

```
⟨ process: Repeat-While rdf:ID = "RWCons"⟩
    ⟨ Repeat-While: whileCond rdf = "Cond" ⟩ ...
                                          ⟨ / Repeat-While: whileCond ⟩
    ⟨ Repeat-While: whileProcess rdf:ID = "whileService" ⟩ ....
        ⟨ process: Perform rdf = "S" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨/Repeat-While: whileProcess⟩
        .....
⟨ /process: Repeat-While ⟩
```

This construct can be modeled by using an atomic query (i.e non-world-altering) task that corresponds to the condition test and recursive call of the subprocess. *RWCons* construct can be represented in CCTR as follows:

$$RWCond \leftarrow ((Cond \otimes S \otimes RWCond) \vee (\neg Cond \otimes stop))$$

In this formula *stop* denotes an atom which is always *true*.

Repeat-Until *Repeat-Until* construct models the iteration that continues until the condition is satisfied. Consider the following generalized repeat-until construct representation in OWL-S:

```
⟨ process: Repeat-Until rdf:ID = "RUCons"⟩
    ⟨ Repeat-Until: untilCondition rdf = "Cond" ⟩ .....
                                        ⟨ / Repeat-Until: untilCondition ⟩
    ⟨ Repeat-Until: untilProcess rdf:ID = "whileService" ⟩ ....
        ⟨ process: Perform rdf = "S" ⟩ ..... ⟨ / process: Perform ⟩
    ⟨/ Repeat-Until: untilProcess⟩
        .....
⟨ /process: Repeat-Until ⟩
```

As in repeat-while, this construct can be modeled by using an atomic query (i.e non-world-altering) task that corresponds to the condition test and recursive call of the subprocess. *RUCons* construct can be represented in CCTR as follows:

$$RUCond \leftarrow (S \otimes (\neg Cond \otimes RUCond) \vee (Cond \otimes stop))$$

In this formula *stop* denotes an atom which is always *true*.

Example 2. The OWL-S representation of the Example 1 was shown in Figure 2. When this representation was mapped to CCTR specification as described above, we have the following service model formula:

$$(((t \vee p) \otimes h) \mid (w \otimes ((g \otimes b) \vee ((\neg g) \otimes s))))$$

5 Reasoning and Verification on Composite Web Services

CCTR provides a framework for the formalization of composite web service and constraints modeling. With this formalization, it is possible to reason about the correctness of the composite model under given constraints. After an OWL-S composite service model is mapped to a CCTR formula through the the control construct mappings given in Section 4, it is possible to do reasoning and verification on the process model under the given set of constraints.

5.1 Verification Under Temporal Constraints

A *temporal constraint* specifies a constraint on the ordering or existence of a service (or a task) with respect to other services. Many of the ordering dependencies can be expressed by control flow constructs, as in OWL-S composite service model. However there may be other dependencies that can not be captured in the control flow specification. For instance, consider the composite process specification: $((a \otimes b) \vee (c \otimes d \otimes e)) \mid (f \otimes g)$ and the constraint *cnst*: "if service c ever executes, it should be done before service f". It is not possible to integrate *cnst* directly in the control flow model.

Transaction Logic can express a rich set of temporal constraints. If *cs* is a formula specifying a composite service model and *cnst* denotes a set of temporal constraints on this model, $cs \wedge cnst$ says that the execution of *cs* should fulfill the constraints *cnst*. Although the expressible set of constraints is wide, we will concentrate on a subset that is common for web services and workflows, as described in [9]:

- Primitive constraints: These are the constraints that directly address the existence of a service as *service s must execute* or *service s must not execute*. They are specified as ∇s and $\neg\nabla s$. ∇s is equivalent to $path \otimes s \otimes path$, where *path* is a special construct of TR which is *true* in all possible execution paths.
- Serial constraints: They are serial conjunctions of positive primitive constraints, such as $\nabla a \otimes \nabla b$.
- Complex constraints: They are the conjunction and disjunction of primitive and/or serial constraints.

Temporal constraint *cnt* defined above is specified as $\neg\nabla c \vee (\nabla c \otimes \nabla e)$. Informally, this formula tells that c is either should not execute or if executed there should be a serial ordering such that c comes before e.

In [9], a verification algorithm is provided such that given a conjunction-free formula *cs* and constraints *cnst*, it checks whether *cs* conforms to *cnst*. If *cs* conforms to *cnst*, *cnst* is compiled into *cs*. In the compilation process, communication primitives in the form of *send/receive* pairs are incorporated into the formula to provide required ordering among services.

For the previous examples where $cs = ((a \otimes b) \vee (c \otimes d \otimes e)) \mid (f \otimes g)$ and $cnst = \neg\nabla c \vee (\nabla c \otimes \nabla e)$. The algorithm finds *cs* and *cnst* to be compatible and complies *cnst* into *cs* resulting in the following specification, cs':

$$cs' = ((a \otimes b) \vee (c \otimes send(\xi) \otimes d \otimes e)) \mid (receive(\xi) \otimes f \otimes g)$$

Example 3. In CCTR, the temporal constraint of the Example 1 is specified as $\neg\nabla b \vee (\nabla h \otimes \nabla b)$. When this constraint is complied into our composite service model given in Example 2, we obtain the following composite model.

$$\begin{aligned}&(((t \vee p) \otimes h) \mid (w \otimes ((\neg g) \otimes s)))\vee \\ &\qquad (((t \vee p) \otimes h \otimes send(\xi)) \mid (w \otimes (g \otimes b \otimes receive(\xi))))\end{aligned}$$

5.2 Verification Under Cost and Resource Constraints

Resource constraints are the constraints on the assignment of resource to a task. In web services context, a resource constraint is a constraint on the selection of service provider among the alternatives. Cost constraint is also highly correlated with resource assignment. A cost constraint is a limitation on total execution cost of process or its subset. Execution cost may have several dimensions such as budget, time or quality. The total values for these dimensions are aggregations of costs of individual services.

In CCTR, resource and cost constraints are represented by a special set of predicates, called *constraint predicate set*, $\mathcal{C}$. The semantics of constraint predicates in $\mathcal{C}$ is specified through relations. If *cs* is a composite service model and *cnst* is a set of resource and cost constraints defined on *cs*, then $cs \wedge cnst$ informally states that the constraints should be fulfilled along the execution of *cs*.

CCTR scheduler checks the correctness of the formula $cs \wedge cnst$ and if *cs* and *cns* are compatible it produces the execution ordering and the service provider assignment on this execution ordering. This process consists of the following steps: transformation of conjuncted formula into a non-conjuncted formula, evaluation of the resulting formula with CCTR proof theory and the evaluation of the constraint by the constraint solver.

Example 4. Assume that the total budget and total duration constraints of Example 1 are represented by the constraint predicates c_1 and c_2, respectively. In the constraint universe, the semantics of these predicates are defined through some relations. Informally, c_1 is mapped to a relation in which the summation of the service providers' prices is below a given value. A similar relation is defined on service providers' service completion time for c_2. Given the service price and service completion information for a set of concrete atomic services, CCTR scheduler checks if there is any service provider assignment along a valid execution and if there is any such assignments generates them.

6 Related Work

There has been intense research on web services. [8], [1], [20] present an overview of the web services and current technology and the standards have been proposed for service modeling, publish, registry and discovery. Some of them can be listed as UDDI [25] for the service registry, SOAP [24] for the communication, OWL-S [11] for semantic modeling and WSDL for service description [27].

There has been several works on the composition of web services. [4] and [18] presents an overview of the web service composition approaches. [26] gives a comparative study of the web service composition languages. Several works presents different approaches and architectures for composition [16], [5], [14], [17], [21], [22], [10]. METEOR-S [16], [17] presents a web service composition architecture in which composition conforms to the given constraints. Another work [28] uses linear programming for solving constraints on composite service. Among the previous work on composite services, the following ones are closer to this work.

[13], [12], [15] proposes a logic-based formalism, *Golog*, for composite service model and verifies some properties of the composite model through Petri Nets. Golog takes its roots from *situation calculus* and provides a strong background for the semantic model. The difference of this work is its ability to support a wide set of constraints, including cost constraints on the overall composite service and do reasoning under these constraints.

[2] maps OWL-S process model to a verification tool in order to do model checking and verification. The emphasis on the verification is common in both works. The major difference is the nature of constraints considered on the process model.

In [23], a mapping from OWL-S to SHOP2 planning language is presented. The mapping schema is close to the mapping considered in this work. However, in this work, the emphasis is on the relationship between OWL-S and CCTR, mapping of the model and verification of the model after the mapping whereas in [23] planning is the main subject of the work.

7 Conclusion

As the Internet grows to include more services, the user requirements will also enhance so that composite services are needed to answer them. As in atomic web services, a semantic model is crucial for composite services. Today's common service ontology, OWL-S provides modeling of composite processes through a set of control constructs.

Besides the semantic model, the correctness of the model under a set of constraints is important as well, for the correct execution and fulfillment of user's requirements. This calls for a verification mechanism through a formalism. In this work, we introduce a logic-based formalism, called Concurrent Transaction Logic (CCTR) and discuss how it can be used for modeling and verification of web services together with OWL-S. CCTR provides a framework for modeling the composite service, specifying constraints on it and verifying the service model under the constraints. The control constructs of an OWL-D description can be mapped to CCTR representation for verification purpose. The verified models can be executed by the profile and grounding descriptions provided by OWL-S.

References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture, and Applications.* Springer Verlag, June 2003.
2. A. Ankolekar, M. Paolucci, and K. Sycara. Towards a formal verification of owl-s process models. In *Fourth International Semantic Web Conference (ISWC 2005)*, 2005.
3. S. Bechhofer and et. al. OWL Web Ontology Language Reference. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004.
4. 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.
5. B. Benatallah, Q. Z. Sheng, and M. Dumas. The Self-Serv environment for web service composition. *IEEE Internet Computing*, 7(1):40–48, 2003.
6. A.J. Bonner. Workflow, transactions, and datalog. In *ACM Symposium on Principles of Database Systems*, Philadelphia, PA, May/June 1999.
7. A.J. 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.
8. F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In *OOPSLA 2001 Workshop on Object-Oriented Web Services*, 2001.
9. H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*, pages 25–33, Seattle, Washington, June 1998.
10. 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.
11. D. Martin and et. al. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/, 2004.
12. 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.
13. S. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, March/April 2001.
14. B. Medjadeh, A. Bouguettaya, and A. K. Elmagarmid. Composing web services on the semantic web. *VLDB Journal*, 12(4), November 2003.

15. S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web*, 2002.
16. K. Verma 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.
17. J. A. Miller 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.
18. 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.
19. P. Senkul, M. Kifer, and Ismail 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.
20. A. Sheth. From semantic search and integration to analytics. In *Semantic Interoperability and Integration*, 2005.
21. E. Sirin, J. Hendler, and Parsia 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.
22. 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.
23. E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau. Htn planning for web service composition using shop2. 1(4):377–396, 2004.
24. Simple Object Access Protocol (SOAP).
http://www.w3c.org/TR/SOAP/, 2003.
25. Universal Description, Discovery and Integration (UDDI).
http://www.uddi.org, 2003.
26. 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.
27. Web Service Description Language.
http://www.w3c.org/TR/wsdl/, 2003.
28. L. Zeng, B. Benatallah, Marlon Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web service composition. In *Twelfth International World Wide Web Conference (WWW)*, 2003.