Formal Verification for CCML Based Web Service Composition

Xiuguo Zhang and Hongbo Liu
School of Information Science and Technology, Dalian Maritime University, Dalian, China

Abstract: Using formal methods to verify web services composition is an important means for ensuring correctness and reliability of web services composition. This study has presented a formal verification method for CCML (Cooperative Composition Modeling Language) based web service composition. First, a mapping mechanism from CCML description of application system to CCS expression is given, then the temporal logic based checking mechanism and automated tool of CWB-NC (Concurrency Workbench of the New Century) is used to address the property verification and service compatibility verification problems of CCML based web service composition. Finally, an application case is presented to show that the formal verification method of CCML based service composition is valid.

Key words: Web service, service composition, formal verification, CCML (cooperative composition modeling language), CCS process algebra

INTRODUCTION

Web services are computational entities that can be used through Internet. They are typically designed to interact with each other in order to compose larger applications. Service composition and corresponding languages describe how services can interact with each other, including the business logic and execution order of the interactions (Liu et al., 2010). Many current service composition languages are non-formal or semi-formal and have no corresponding formal supporting tools (Yang and Qin, 2010; Zhang and Zhan, 2006). So these service composition languages are error-prone and difficult to check and verify correctness of service composition (Yin et al., 2010).

Formal method is an effective way for modeling and verifying software system. At present, the main formal methods of modeling and verifying web service composition are Petri Net based method (Li et al., 2011; Fang et al., 2009), process algebra based method (Deng et al., 2006; Yang et al., 2008) and finite state machine based method (Foster et al., 2006). Petri Net based method focus on describing behavior process inside web service and adapts to verify correctness of composite service behaviors (Yan et al., 2007). Process algebra based method focus on describing behavior process outside web service and adapts to verify interaction correctness among web services (Deng et al., 2006; Gu and Lu, 2006). Finite state machine based method describes service behaviors from the view of state transition and message passing. But it is difficult to describe behaviors of sophisticated process (Foster et al., 2006).

CCML (Cooperative Composition Modeling Language) (Zhang and Zhang, 2006) is a kind of service composition modeling language which is designed by ourselves to address behavior composition problems of interacting web services. In CCML, we define four kinds of service interaction events which enable services to communicate with each other. In this study, we present a formal verification method to solve property verification and service compatibility verification problems of CCML based service composition.

INTRODUCTION TO CCML

The basic structure of CCML is as follows:

```
service service-name{
  //describing service operations
  operation operation-name{
    //describing ports of input and output
    input:
    ForType port <name of port1>, <name of port2>, ...
    output:
    ForType port <name of port1>, <name of port2>, ...
  }
  ...
  //defining local variables
  VarType <name of variable1>, <name of variable2>, ...

  //defining instances of sub-service
  Instance <name of sub-service> <instance name of sub-service>

  //defining channel connection
  Channel <name of channel1>, <name of channel2>, ...

  //describing service composition, service composition is based on Control Construct and service interaction events
}
```

Corresponding Author: Hongbo Liu, School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China

1692
In CCML, we define four kinds of service interaction events which enable services to communicate with each other.

**Control constructs and events:** We define several control constructs which are Sequence, Condition, Loop and Parallel to represent an implicit potential communications path among service operations. As to each control construct, we describe it with the following format:

1) **Sequence construct**
   - Operation(op1);
   - Operation(op2);
   ·······
   - Operation(opn);

2) **Condition construct**
   - switch{
     - case <boolExpr1>: do Operation(op1);
     - case <boolExpr2>: do Operation(op2);
     ·······
     - default Operation(opn);
   }

3) **Loop construct**
   - while <boolExpr> do{
     - Operation(op1);
   }

4) **Parallel construct**
   - parallel{
     - Operation(op1);
     - Operation(op2);
     ·······
     - Operation(opn);
   }

We also provide several basic control events to initialize, execute and terminate a service or service operation:

- **Init():** This event is used to initialize a service and prepare the necessary resources at runtime
- **Operation(op1):** This event is used to execute a service operation op1 of a initialized service
- **Stop(op1):** This event unconditionally terminates a running service operation op1 and releases occupied system resources

**Port binding events:** In CCML, a composite service is composed by sub-services and their interconnections. Port binding events are used to indicate a port connection inside a composed service. There are three cases of port binding. The first one is called In-In binding, i.e., data are transmitted from an input port of service interface to an input port of sub-service. The second one is called Out-In binding, i.e., data are transmitted from an output port of a sub-service interface to an input port of another sub-service. The last one is called Out-Out binding, i.e., data are transmitted from an output port of a sub-service interface to an output port of service.

In this study, we define three events to represent the three port binding relations:

- **InIn(S<Pin>, S<Pin>):** This event is used to create an In-In binding
- **OutIn(S<Pout>, S<Pin>):** This event is used to create an Out-In binding
- **OutOut(S<Pout>, S<Pout>):** This event is used to create an Out-Out binding

where, S<Pin> represents an input port of a service and S<Pout> represents an output port of a service.

**Variable binding events:** Variable binding events are used to indicate that a service writes to a local variable inside a service through one of its output ports or a service reads data from a local variable through one of its input ports. In this study, we use the following events to represent them:

- **Vwrite (V, S<Pout>):** This event represents a service writes to a local variable V via S<Pout> which is an output port of this service
- **Vread (S<Pin>, V):** This event represents a service reads data from a local variable V via S<Pin> which is an input port of this service

**Channel transaction events:** Channel transaction events are used to indicate data flowing from a service to another service over a channel. We define three channel transaction events in this study:

- **Create channel (C, S<P1>, S<P2>):** This event is used to create a channel connection C between service port S<P1> and S<P2>
- **Read channel (S<Pin>, C):** This event represents a service reads input data from FIFO queue of channel C via input port S<Pin> of this service
- **Write channel (C, S<Pout>):** This event represents a service writes output results into FIFO queue of channel C via output port S<Pout> of this service

**MAPPING CCML TO CCS PROCESS ALGEBRA**

To ensure correctness of CCML based web services composition, we use CCS process algebra (Milner, 1989) to verify service composition models described with CCML. CCS process algebra provides a high expressive power and is enough to fulfill formal verification of CCML based service composition. First, mapping CCML description of service composition to CCS expression, then using temporal logic based checking mechanism and
automated tool of CWB-NC (Concurrency Workbench of the New Century) (Cleaveland et al., 2000) to address the property verification and service compatibility verification problems of CCML based service composition.

Mapping basic elements of CCML to CCS: CCML has some basic elements, including services, composite services, channels, service ports, service operations and local variables. Table 1 shows the relations between basic elements of CCML and CCS.

In CCML, channel is viewed as a kind of FIFO container of data tokens which represents data flow among services. These channels are equal to messages received and sent by CCS process. So CCML channel is mapped to CCS message. In CCML, a service is composed of operations and an operation comprises several input and output ports. So input/output port of a service is mapped to reception/emission action of CCS. For example, in CCML, a port p is defined as:

```
input:
  string port p;
  which is mapped to CCS and represented as, string p(?)
```

Finally, local variable of CCML is mapped to CCS message.

Mapping interaction events of CCML into CCS representations: Four kinds of interaction events of CCML are mapped to related CCS representations, as showed in Table 2.

In CCML, writing operation on channel is viewed as a composite action of InIn() / OutOut() , CreateChannel() and WriteChannel() which is mapped to a writing action (a;x) of CCS. Reading operation on channel is viewed as a composite action of InIn() / OutOut() , CreateChannel() and ReadChannel() which is mapped to a reading action (a;x) of CCS. To interaction events those are not included in composite action, such as Init(), Operation(), Stop(), Vread() and Vwrite() are not taken into account in mapping.

Mapping control constructs of CCML into CCS representations: In CCML, there are four kinds of control constructs which are sequence, selection, loop, parallel. Table 3 shows the mapping from Control Constructs of CCML to CCS Representations:

<table>
<thead>
<tr>
<th>Table 1: Relations between basic elements of CCML and CCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic elements of CCML</td>
</tr>
<tr>
<td>service</td>
</tr>
<tr>
<td>composite service</td>
</tr>
<tr>
<td>channel</td>
</tr>
<tr>
<td>service operation</td>
</tr>
<tr>
<td>port</td>
</tr>
<tr>
<td>variable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Mapping interaction events of CCML into CCS representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction events of CCML</td>
</tr>
<tr>
<td>Init()</td>
</tr>
<tr>
<td>Operation(op1)</td>
</tr>
<tr>
<td>Stop(op1)</td>
</tr>
<tr>
<td>Vread(S&lt;Pn&gt;, V)</td>
</tr>
<tr>
<td>Vwrite(V, S&lt;Pn&gt;)</td>
</tr>
<tr>
<td>InIn(S&lt;Pn&gt;, S&lt;Pn&gt;)</td>
</tr>
<tr>
<td>OutOut(S&lt;Pn&gt;, S&lt;Pn&gt;)</td>
</tr>
<tr>
<td>CreateChannel(C, S&lt;Pn&gt;, S&lt;Pn&gt;)</td>
</tr>
<tr>
<td>ReadChannel(S&lt;Pn&gt;, S&lt;Pn&gt;)</td>
</tr>
<tr>
<td>WriteChannel(C, S&lt;Pn&gt;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3: Mapping control constructs of CCML into CCS representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control constructs of CCML</td>
</tr>
<tr>
<td>Sequence construct</td>
</tr>
<tr>
<td>Selection construct</td>
</tr>
<tr>
<td>Loop construction</td>
</tr>
<tr>
<td>Parallel construct</td>
</tr>
</tbody>
</table>

- Selection construct of CCML is mapped into CCS choice construct which is represented as, Sw = .op1 +.op2+...+.opn. Where, is an internal action which is used to decide which case will be executed
- Loop construction of CCML is mapped into CCS recursive construct which is represented as, op1(x) = op1. Where, is a sequence of data, op1() represents emission or reception action of operation op1
- Parallel construct of CCML is mapped into CCS parallel construct which is represented as, Par = op1| op2|...| opn

VERIFICATION METHOD OF CCML BASED SERVICE COMPOSITION

In CCML, we address two kinds of verification problems which are property verification and compatibility verification. Related verification mechanism can reason about the system's behavior and verify the model's properties such as deadlock and reachability after modeling the system.

Property verification of CCML based service composition: Property verification is used to verify whether a composition satisfies the desired properties or not, for example, the system should never reach a "bad state", the desired state will be arise sooner or later, etc.
Fig. 1: Verification process of model properties

Figure 1 presents the property verification process in CCML. First, we map CCML description of web service composition to CCS expression, then we use Modal $\mu$-calculus (Bradfield and Stirling, 2001) to describe temporal logic properties of this composition model, finally, automated tool of CWB-NC (Concurrency Workbench of the New Century) is used to address the property verification problems of CCML based Composition model.

Using $\mu$-calculus model-checker provided by CWB-NC, the first step is creating .mu file.

In .mu file, deadlock property is represented as:
\[
\text{prop can-deadlock} = \min X = [-] \text{ff } \lor \iff X
\]
where, can-deadlock is a safety property. When it is true a system may reach a deadlocked state.

Reachability is represented as follows in $\mu$-calculus:
\[
\text{prop can-response} = \min X = (\text{\textless Refuse\textgreater }tt \lor \text{\textless Appointment\textgreater }tt) \lor \iff X
\]
where, can-response is a liveness property of system which means a Refuse action or a Appointment action will be executed after a finite steps.

In addition, we take into account a system property UpdBeforeReqIn that is the emission of message UpdWO is prior to reception of message ReqInv. This property is another safety property, represented as follows in $\mu$-calculus:
\[
\text{prop UpdBeforeReqIn} = \text{AG} (\text{not}[\text{UpdWO}]tt) \lor \text{EF} (\text{ReqInv} \lor tt))
\]

We use $\mu$-calculus model-checker provided by CWB-NC to verify the above properties, the verification command is as follows:

- \text{chk SERVICE can-deadlock}
- \text{chk SERVICE can-response}
- \text{chk SERVICE UpdBeforeReqIn}

The verification result is TRUE or FALSE which means the system satisfies the property or not.

**Compatibility verification of services:** Compatibility of web services states the fitness of service peers that interact with each other. It covers both static properties and dynamic behavior of web services. This requires reasoning on behavior features of web services.

Let \( Q \) be the set of services, \( q \in Q, q' \in Q \) where, \( 1 \leq n \leq N \), \( N \) is set of positive integers.

**Definition 1 (weak transitions):** Let, \( q \in Q, q' \in Q, \tau \) is transition label for activity which is not externally visible, if:
\[
- q \xrightarrow{\tau} q' \text{ iff } q = q_0 \rightarrow^* q_1 \vdash \cdots \vdash q_n = q'= q', n > 0
- q \xrightarrow{\tau} q' \text{ if } q = q' \vdash q_0 \vdash \cdots \vdash q_n = q' = q'(x \tau)
\]
then, \( q \xrightarrow{\tau} q' \) is a weak transition

**Definition 2 (observational equivalence):** Let, the relation \( S \) is a weak bisimulation relation if Whenever \( q \vdash S q' \) then:
\[
- \text{if } q \xrightarrow{\tau} q', \text{ for some } q'', \text{ implies } q'' \text{ and } q' \vdash S q''
- \text{if } q \xrightarrow{\tau} q', \text{ for some } q'' \text{ implies } q'' \vdash S q''
\]
then, \( q \) and \( q' \) is observationally equivalent, or weakly bisimulation equivalent that is, for an external observer, it is not possible to distinguish the behavior of \( q \) and \( q' \).

**Definition 3 (behavior compatibility):** Two services \( P \) and \( Q \) are compatible if they have opposite behaviors, i.e., \( P \) is Observationally Equivalent to \( Q \).

For example, let ‘\( ! \)’ denotes message emission, \( \text{emissionP(P)} \) denotes the emission message sequence of service \( P \), \( \text{receptionQ(Q)} \) denotes the reception message sequence of service \( Q \).

Let \( P = !\text{Req.Receive.nil} \) \( Q = \text{Req}!\text{Receive.nil} \), it’s clear that, \( \text{emissionP(P)} = \text{receptionQ(Q)} \) and \( \text{emissionQ(Q)} = \text{receptionP(P)} \). When \( P \) and \( Q \) execute a reception/emission action, respectively that is then, \( P!\text{ReqP} \), \( Q\text{ReqQ} \), \( P' = \text{Receive.nil}\) \( Q' = !\text{Receive.nil} \) it’s clear \( \text{emissionP(P)} = \text{receptionQ(Q)} \) and \( \text{emissionQ(Q)} = \text{receptionP(P)} \). When \( P \) and \( Q \) execute a reception/emission action, respectively that is \( P!\text{Receive.nil}, Q'!\text{Receive.nil}\) and \( Q' \) will be nil. It’s clear that, behaviors of \( P \) and \( Q \) is opposite. So they are behavior compatible.
CASE STUDY: CORRECTNESS VERIFICATION OF CCML BASED WEB SERVICE COMPOSITION

Imagine a Vehicle Repair Management system as shown in Fig. 2. This system is composed of five services which are Client Service, Customer Service, WorkOrder Service, Inventory Service and Scheduling Service.

The basic business process of this Vehicle Repair Management system is as follows:

- A Client Service sends repairing request to Customer Service through channel ‘ReceiveTel’. Customer Service asks about the details of car through channel ‘AskInfo’. Client Service provides the details through channel ‘ProInfo’
- Customer Service receives repairing request and asks WorkOrder Service to create work order through channel ‘ReqCreWO’
- WorkOrder Service sends request to Inventory Service for querying stock of parts needed to repair car through channel ‘LookUp’
- Inventory Service returns the result of whether or not the stock is satisfied to WorkOrder Service through channel ‘InRep’ or ‘InRef’
- WorkOrder Service replies to Customer Service whether or not the work order can be created through channel ‘WoRep’ or ‘WoRef’. If the work order can be created, WorkOrder Service will send the work order to Scheduling Service and notifies Scheduling Service to prepare repairing workshop through channel ‘ReqPre’
- Customer Service notifies Client Service whether or not his car can be repaired according to the reply from WorkOrder Service through channel ‘Appoint’ or ‘Refuse’
- Client Service sends acknowledgment message to Customer Service through channel ‘ClientCon’ and sends arrival message of his car to Customer Service through channel ‘Arrival’
- When the car comes, Customer Service notifies Scheduling Service to repair it through channel ‘ReqRepair’
- Scheduling Service sends request to Inventory Service for parts needed to repair car through channel ‘ReqArti’
- Inventory Service sends the needed parts to Scheduling Service through channel ‘DeliverAr’
- When the repair finishes, Scheduling Service notifies Customer Service through channel ‘End’, meanwhile notifies WorkOrder Service through channel ‘UpdWO’
- Customer Service receives the repair finish message, asks WorkOrder Service to compute repair fee and send receipt to Customer Service through channel ‘ReqInv’
- WorkOrder Service computes repair fee and sends receipt to Customer Service through channel ‘Send’
- Customer Service notifies Client Service the repair is finished through channel ‘CompRep’
- Client Service takes the car and receipt away and the repair process finishes

Property verification of vehicle repair management system: First, describing this system in CCML, then mapping CCML description to CCS Process Algebra.

Fig. 2: A Vehicle repair management system
representation using mapping methods introduced. The corresponding .csl file is as follows:

- proc Client = 'ReceiveTel, AskInfo, ProInfo, (Assign, WorkOrder, CompRep, Client, Reject, Client)
- proc Inventory = 'LookUp, 'InRef, 'Inventory, 'InRep, 'ReqArti, 'DeliverAr, 'Inventory
- proc SERVICE = (Client) Customer, WorkOrder, Inventory, Scheduling, (ReceiveTel, AskInfo, ProInfo, ReqCreWO, LookUp, InRep, InRef, WoRep, WoRef, Reject, Assign, ClientCon, ReqPre, Arrival, ReqRepair, CompRep, End, UpdWO, ReqInv, Send, ReqArti, DeliverAr)

- Creating the .mu property description file:

  prop can-deadlock = min X = [-]IF/<>-X
  prop can-response = min Y = ([Refuse] tt/[(Assign) tt])/ <> Y

- Creating the property verification file service.csw:

  - load service.csl
  - load service.mu
  - chk SERVICE can-deadlock
  - chk SERVICE can-response
  - chk SERVICE UpdBeforeReqIn

- Using µ-calculus model-checker provided by CWB-NC to verify can-deadlock, can-response and UpdBeforeReqIn property. Figure 3 shows the verification result of can-deadlock property

The result of can-deadlock property is FALSE which shows this system can not reach a deadlock state. For can-response property, the verification result is TRUE and for UpdBeforeReqIn property, the verification result is TRUE which accord to the real system.

Compatibility verification of vehicle repair management system: In this case, we only verify the behavior compatibility between Client Service and Customer Service, the same way for other services:

- Getting the behavior description of Client Service and Customer Service:

![Concurrency Workbench for CCS](image)

Fig. 3: Verification Result of can-deadlock Property
Fig. 4: Verification Result of Behavior Compatibility between Client Service and Customer Service

proc Customer = ReceiveTel.'AskInfo.ProInfo.'ReqCreWO.(WoRef.Refuse.nil+WoRep.'Appoint.Client 'Con.Arrival.'ReqRepair.End.'ReqInv.Send.'CompleteRep.nil)

- Getting the opposite behavior description of Customer Service:


- Using verification command “eq -S obseq Client reverseCustomer” to verify if Client Service and reverseCustomer Service is Observationally Equivalent

Figure 4 shows the verification result of behavior compatibility between Client Service and Customer Service. The result is TRUE which shows Client Service and Customer Service is compatible.

STATE OF THE ART

At present, verification method of web services composition mainly focus on two aspects: one is verifying property of composition model including safety and liveness property which belongs to the area of correctness, another is reasoning on system behavior including analyzing compatibility of service behavior, analyzing consistency between system behavior and requirement behavior.

Many researches address the property verification problems. For example, Zhang et al. (2011) adopt timed PI calculus to model service behaviors and interactions in a formal way and ensure the correctness of services composition. Wang et al. (2010) use Temporal Logic of Actions (TLA) to model Web services, composition patterns and users' requests. The correctness of composition result can be verified by checking whether the resulted TLA formula implements a user's request. Salaun et al. (2004) adopts CWB-NC model-checker to verify safety and liveness property of system model. Yang et al. (2008) present a web service choreography description language CDL and provide a set of translation rules from CDL to Promela which allows the user to model and check choreographies in SPIN. Liao et al. (2005) adopt MWB (Mobility Workbench) tools to verify correctness of web service composition including type checking, behavior integrity checking and deadlock checking, etc.

As for service compatibility, Li et al. (2011) use colored Petri nets (PNs) to verify the behavioral compatibility among well-structured Web services to ensure the correctness of the whole composition. Xiong et al. (2010) model multiple Web service interaction with a Petri net. A policy based on appending additional information channels is proposed to address the behavioral compatibility problem (Xiong et al., 2010). Foster et al. (2004) adopt Finite State Process notation (FSP) to describe web service composition and verify
composition compatibility from interface compatibility, safety compatibility and liveness compatibility. Andre et al. (2005) use LOTOS/CADP tools to verify service compatibility from interface and behavior levels. Deng et al. (2007) propose a pi-calculus based method to qualitatively determine and quantitatively compute behavioral compatibility. With the help of operational and transitional semantics of pi-calculus they can determine qualitatively whether two services are behavioral compatible. After that they proposes an algorithm based on the Expansion Law to compute the compatibility degree between services quantitatively.

As for consistency verification between system behavior and requirement behavior, Salaun et al. (2004) propose a simple and effective reverse method to verify observational equivalence of CCS process algebra based system. In fact, observational equivalence is the consistency between the opposite requirement behavior and system behaviour (Wang et al., 2010). Wang et al. (2009) research semantic consistency verification of web service composition. They describe web service composition using semantic Petri nets and propose verification algorithms of semantic consistency including function consistency, behavior consistency and QoS consistency.

In this study, we tackle the property verification and service compatibility verification problems of CCML based service composition, further to ensure the correctness of CCML based service composition.

CONCLUSIONS

In this study, we present an approach to verifying CCML based service composition using CCS process algebra and related checking mechanisms. We provide a set of translation rules for mapping CCML description to CCS expression which allows users to verify CCML based service composition using temporal logic based checking mechanism and automated tool of CWB-NC. We also present property verification and service compatibility verification method for CCML based service composition.

The future study is to develop an automated converting tool for translating CCML description to CCS expression and to verify other important properties including strong bisimulation equivalence and congruence equivalence of web services in CCML based service composition.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China (Grant No.60873054, 60973013, 61073056, 61073134), the Fundamental Research Funds for the Central Universities (Grant No.2011JC006) and Dalian Science and Technology Fund (Grant No.2010J21DW006).

REFERENCES


