Binary Activity Chain Modes for Compositional Web Service and Its Compliance Verification

Bo Chen Chungui Li Qixian Cai
Department of Computer Engineering, Guangxi University of Technology, Liuzhou, 545006, China
cb31@sina.com lcg6881@tom.com cqx_gx@163.com

***Abstract*—requirement-driven behavior verification for compositional Web service is one of hot research issues for Web computing. Modeling and analysis of the behavior requirements of compositional Web service plays an important role in behavioral verification. Traditional methods are expressing requirements as LTL like logic specifications which are based on activities or as MSC like graph forms which are based scenarios. In this paper, we propose the concept of behavior specification based on activity chain in which its atomic granularity is between activity and scenario. Four behavioral modes such as chain existence mode, chain absence mode, chain precondition mode and chain response mode are adopted to express usual requirement specifications. Encode them on Labeled Transition System LTS and then give them operation semantics. Check compositional Web services based on LTS corresponds with behavior modes or not. Give the sufficient, necessary condition and algorithm for checking.**

***Index Terms*—Specification, Stateful Web Services, Composite Web Services, Model checking**

I. INTRODUCTION

A fundamental goal of web services is to combine today's simple Web services into more complex ones in order to achieve more sophisticated application purposes. The composite service adds two dimensions by comparison to the simple ones; they are stateful and they obey to an operational behavior. This raises many theoretical and practical issues which are part of ongoing research [1]. Recently, the works on web service verification are mainly focus on the following three issues 1) whether communication activities of compositional service accord with specification; their interactions are compatible? A given service can be replaced by another? [1], [2], [3]. 2)whether the control and data flows of compositional service processes are correct? and whether the subservices comply with the constraint rules among them? [4], [5], [6]. 3) whether a compositional service is compliant with specific requirements of the user (requirement-driven compliance verification)?

The ultimate motivation of web service composition is to offer satisfactory behavioral functions for users. It is important to study concise methods to express the requirements for behavior of compositional service and check whether the behavior is compliant with the requirement of user after checking that service process is correct and communication is available. Some works on this issue have been published. Pistore [12] expressed the goals and requirements of different roles in compositional service with formal Tropos language. Also the internal constraints and external dependencies to implant these goals and requirements have been formally presented. But the requirements are only for some component of specific roles, nor a whole behavior requirements of an user for a compositional service. Furthermore, the expressions of such requirements are in some LTL like forms not in concise manners. Rouached [13] expressed the time, casual and results of events occur in compositional web service with event calculus and attributes of behavior are expressed with first order logic. But a requirement expressed in such manner is in essence relation among some single events not relation among certain event sequences and its expression manner is more abstract. More existed works[7-13]generally expressed behavioral requirements in two methods such that one is in LTL,CTL like temporal logic specifications and another is in MSC, UML like graphic specification. The former's basic element is activity. The latter's basic element is scenario. The specifications describe some relation among activities or scenarios. In case of compositional web service, requirements for service behavior are often demands for composite behavior that are temporal relations between activity chains which belong to different subservice component. Since the direct object described by LTL like logic and MSC like graphic language is activity and scenario respectively, it is not suitable for LTL and MSC to direct describe such temporal relations based on activity chain which its granularity is between activity and scenario. It is nature to consider to transform the temporal relation based on activity chain to the temporal relation based on activity and then to express indirectly such behavioral requirement with LTL like logic. However, such a behavioral requirement contains two hierarchy temporal relations that first is the temporal relation between two activity chains and second is the chain order of activities in a specific activity chain. So obtained LTL formula after transformation will be very complicated. Not only is this formula difficult to read and understand, it is even more difficult to write correctly without some expertise in the idioms of specification language [15]. It is still a problem

* Supported by Guangxi Sci. & Tec. Program under grant of 0992006-13, Guangxi Nature Science foundation under grant of 0481016

doi:10.4304/jsw.4.8.907-914

to be explored how to suitably express the behavioral requirement based on activity chain in certain concise manner.

In current existed work on requirement-driven compliance verification, the principal approach is to translate a service behavior (BPEL process) into a mathematically well-founded model, considering only the semantic of elements that are relevant for the property to be verified. Then, model checking methods can be applied to the formal representation of the composite service behavior [1]. The behavior requirements to be checked are temporal relations based on activity or scenario. Fu [7] modeled the asynchronous communication of partners in compositional web service as a guarded mealy automaton. The global message chain observed by virtual observer is as the session model of compositional web service. In the case of finite input message queue, checked whether the session behaviors comply with the communication specification which is a LTL formula based on message. In his specification, the basic element is activity which sends or receives a message. Betin-Can [8] designed an interface protocol as communication contract expressed with guarded automata and checked whether a service complies with the contract by Java PathFinder. But the contract is only the specification of subservice not that of global compositional service. Mongiello [9] abstracted a BPEL process as an execution chain and verified the chain against a specification expressed in CTL. The basic element of a specification was still activity. Foster [10] expressed behavior requirements with MSC and the obligation of service with fluentLTL and verified compliance of a compositional service in LTS. The basic element of a specification is scenario or activity with effect. Aalst [11] intercepted the flow of SOAP message of a compositional web service and translated them into Petri net as behavior model. The corresponding BPEL process was also translated into Petri net as a specification. The service implementation compliance verification was carried out in accordance with the specification which was a Petri net. Note that the basic elements of all specifications in above works are activity, scenario or even a state transition system. Their granularities are too small or too large to suitable to express such behavior requirements which granularity is middle level that is activity chain. How to express such behavior requirement in certain concise specification manner and how to verify that a compositional web service is compliant with such a specification or not? It is still an issue to be explored.

In this paper, we focus on the expression of binary behavior requirement based on activity chain and attempt to formally define such behavior specifications. Then we give the methods on checking the compliance of compositional web service against such specifications.

II. The Basic Concept of Activity Chain in Service and Example

A. The Definition of Activity Chain

Definition2.1. the activities of a compositional web service is inductively defined as below

$WS= \{ws_1, ws_2 ,\ldots,ws_k, ws_{orch}\}$is a set of web service instance names, Where ws_{orch} is an orchestration engine.

$O_{ws}= \{o_{ws}|o_{ws}=$ op [? m] or op [! m]$\}$ is the set of operations in the port type of service ws, where op is a operation name, m is a message name, ?m means receiving a message m, !m means sending a message m.

$A_{ws}= \{a_{ws}|a_{ws}=$ receive[o]ws_{orch} or reply[o]$ws_{orch}\}$ is a set of basic activities of service ws. Activities are classified in two categories that one is receive a request for operation o of ws from service wsorch and another is answer to the request for operation o.

$A_{orch}= \{a_{orch}|a_{orch}=$ receive[o]ws or reply[o] ws or invoke $[o_{ws}]$ $\}$ is the set of basic activities of service wsorch. invoke $[o_{ws}]$ means wsorch invokes an operation ows of service ws.

$O=\cup \{o_{ws}|ws\in WS\}$ is the set of operations of compositional web service WS.

$Act=\cup \{ a_{ws}|$ aws$\in A_{ws}$, ws$\in$WS$\}$ is the set of activities of compositional web service WS.

In the definition above, every operation of a service must abide by the specification of WSDL. It belongs to one of four categories that is notification, solicit, request-response and solicit- response expressed in definition as o[!m],o[?m],o[!m, ?m],o[?m, !m] respectively. Since the so-called two-way operations of two latter categories are easily transformed into one-way operations of two former categories, we adopt only one-way operations such as o [! m] and o [? m] in this paper and let o, a represents symbols of operation and activity respectively in case of unambiguity.

Definition2.2.an activity chain of a compositional web service with length n, $C=<a_1, \ldots, a_n>$, $a_i\in Act,1\le i\le n,$, is a tuple of finite activities occurred one after another in an execution of service.

Specially, if one infinite activity chain is composed of all activities occurred sequently in one execution of service, it is called a trace of service. Denote itσ.

B. Binary Activity Chain ModesDefinitions

Definition2.3. Let C=<a1,… , an> an activity chain of service WS with length n, $\sigma=<\sigma_1,\ldots,\sigma_k,\ldots>$, $\sigma_i\in Act,1\le i$, is a trace of WS. If there is a finite subchain of σ with length n, $\sigma^i= <\sigma_{i1},\ldots,\sigma_{in}>$, that $\sigma_{ij}=a_j$, $1\le j\le n$, then call C occurs in σ .

Definition2.4. Let C an activity chain of service WS with length n and a is an activity of WS. If for any trace σ that C occurs in σ , a must occur in σ precedence of C, then C and a satisfy chain precondition mode and is written a ***C_PR*** C.

Definition2.5. Let C an activity chain of service WS with length n and a is an activity of WS. If a occurs in traceσ, it will lead to C occur in σ after a occurs in σ. Then C and a satisfy chain response mode and is written a ***C_RE C.***

C. An Example

Flight and Hotel are two existed web services, which provide separately flight and hotel booking service for the Client. Travel agency F_H is the compositional service orchestrator, which provides integrated service for client. F_H is responsible for invocation Flight operation and Hotel operation, and interact with client at the same time to provide integrity service process. Figure 2.1 describes an interactions scenario of compositional service. F_H accepts the request from the client, and invokes Flight subservice and Hotel subservice concurrently, returns sorted flights list, lodging suggestions, then provides appointed flight and lodging plan according to client's request, and waiting for client's confirmation. At last, F_H provides the client with services (the ticket and lodging order). We omit some communications between F_H and Hotel for the sake of simplicity. Interactions of services are described by MSC like graph language. The difference here is that we use activity instead of message to label the transition of MSC. $\xrightarrow{a}$ denotes the execution of the activity a, the label a on the row denotes an activity of service, the arrow denotes control flow direction when executing activities in interaction process.

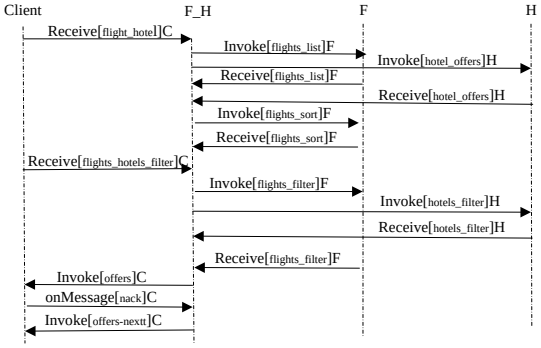

Figure 2.1. An execution scenario of F_H

Before using the compositional service, the user need validate whether compositional service is compliant with requirement. The requirement for behavior usually only involves local activities or activity chains of compositional service, which present as temporal relations of activities or activity chains. The following are two types of behavior requirements in example.

TABLE 2.1. THE BINARY BEHAVIOR REQUIREMENTS BASED ON ACTIVITY CHAINS IN EXAMPLE

requirements	Behavior mode	activity chains
R_2.(3)	a ***C-PR*** C1	a=receive[hotel_offers]H,C1=<receive[flights_hotels_filter]C, invoke [flights_filter]F>
R_2.(4)	a ***C-RE*** C1	a =invoke[nack]C ,C1= <invoke[offers]C,onMessage[nack]C>

R1:(1) The service must provide optional flights sorted by price after have received the client request.

(2) The service must not provide tickets and lodging orders after the user refusing to acknowledge finally.

R2:(3)Hotel candidate offers provision must be precondition of providing flights filter.

(4) If client reply nack for offers, then service will not provide final service.

R1.(1) is a liveness property that can be expressed as: G(receive[flight_hotel]C→true U invoke [flights_sort]F) which is based on activity receive[flight_hotel]C and activity invoke[flights_sort]F. Similarly, R1.(2) is a safety property based on activities.

R2 are behavior requirements based on activity chains.

The table 2.1 lists R_2 bebavior requirements in example.

III THE SEMANTICS OF BINARY ACTIVITY CHAIN MODES

In order to verify the compliance of compositional web service, it is need to give operation semantics for activity chain modes. The labeled transition system (LTS) is widely used to describe the dynamic semantic of distributed concurrent system [16]. In this section, we encode the activity chain modes presented in section 2 into LTS and give these modes the precise interpretation

Definition3.1. An LTS is a tuple L=(S,A,→, s),where S is the set of finite states. A=αL ⊆Act is the set of finite activities. →⊆S×Aτ×S is a transition relation. Aτ=A {τ}. s is initial state and τ is internal activity that is invisible to extern.

When L executes an activity a, a∈Aτ, (s, a, s')∈→, then it may become L', L'=(S, A,→, s'). Denote it $L \xrightarrow{a} L'$, iff $s \xrightarrow{a} s'$, here, $s \xrightarrow{a} s'$ is the same mean that of (s, a, s')∈→.

Definition3.2. Let LTS L_1, L_2 are two LTS. The parallel of two LTS is the LTS L denoted as $L=L_1\|L_2$. The rules of parallel operation of two LTS are listed below

$$(1)\frac{L_1 \xrightarrow{a} L_1'}{L_1 \| L_2 \xrightarrow{a} L_1' \| L_2}, a \notin \alpha L_2 \quad (2)\frac{L_2 \xrightarrow{a} L_2'}{L_1 \| L_2 \xrightarrow{a} L_1 \| L_2'}, a \notin \alpha L_1$$

$$(3)\frac{L_1 \xrightarrow{a} L_1', L_2 \xrightarrow{a} L_2'}{L_1 \| L_2 \xrightarrow{a} L_1' \| L_2'}, a \in \alpha L_1 \cap \alpha L_2$$

Definition3.3. Let L=<S, A,→, s_0>,A=αL, is a LTS. ρ=$s_0a_1s_1a_2$ s_2…is an infinite or finite alternating chain of states and activity labels, where s_i∈S, i≥0, a_j∈A,j≥1, $s_{i-1} \xrightarrow{a} s_i$, i≥1. ρ is called an execution of L. σ=<a_1,a_2…> is called the trace corresponding to ρ.

Definition3.4. Let L=<S,A,→,s_0>,A=αL, is a LTS. s∈S, a∈αL∪{τ}, Post(s, a)={s'| $s \xrightarrow{a} s'$} is the set of direct successive states of s related to activity a. Post(s) = $\cup_{a\in\alpha L\cup\{\tau\}}$Post(s, a) is the set of direct successive states of s

A state s is called termination state of L when Post(s)= Φ. An execution of L is called finite termination iff after finite steps of execution of L, ρ=$s_0a_1s_1…s_n$ and Post (s_n) =Φ.

In this paper, a web service is expressed as a LTS and a compositional web service is expressed as the parallel of finite LTSs that is $L=L_1\|L_2\|…\|L_k$, Li, 1≤i≤k,

represents a subservice. L is a dynamic behavior model of compositional web service. In order to facilitate users to express the behavior requirements based on activity chains, such requirements are expressed as two binary behavior modes in section II. However, the exact meaning of every mode, that is its operation semantic, is still needed to be interpreted by LTS. The mapping rules from behavior modes to LTS have been listed in figure 3.1 below. LTSs (3),(4)in figure 3.1 are extended with accepting states and accepting activity. The detailed interpretations are in section IV.

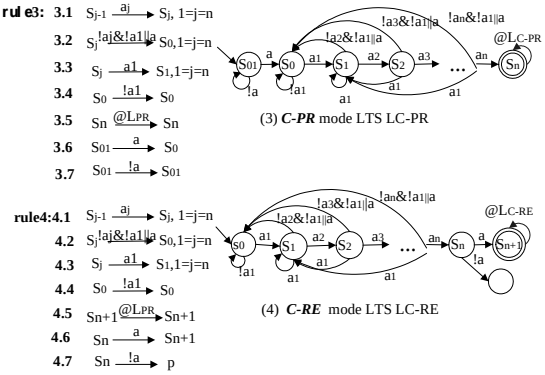

Figure 3.1. mapping rules of activity chains and LTSs

IV COMPLIANCE VERIFICATION WITH ACTIVITY CHAIN MODES

Compliance verification is to check whether every execution of specific compositional web service complies with the activity chain mode. In general, a specific LTS presents a ceaseless reactive system which its executive path is infinite. Thus if it goes into a terminate state, that is Post(s) =Φ, s must be its deadlock state. However, a LTS represents a web service in this paper and its normal execution may be infinite or finite normal conclude. If a state s is a terminate state, it may be possible a deadlock state or possible normal finite conclusion state. Compliance verification must go on in normal execution of web service. So it is necessary to exclude the deadlock situation from a web service first

A. The Extension of LTS and Exclusion of Deadlock States

Assume the LTS $L_W=L_1||L_2||\ldots||L_k$ is a compositional web service is terminated in state $s=(s_1,\ldots,s_k)$, $s_i, 1\le i\le k$,is corresponding subservice state of the service, post(s)=Φ.

Definition4.1. If a state s is a termination state of a compositional web service, then

(1) s is a normal conclusion state(also called accepting state) iff $\forall i.\ 1\le i\le k.\ Post(s_i)=\Phi$ and s_i is L_i normal conclusion state.

(2) s is a deadlock state iff $\exists i.\ 1\le i\le n.\ Post(s_i)\neq\Phi$.

To pick out deadlock state, we extend the LTS model L below.

1. If s_i is an accepting state of L_i, add a self-loop transition with activity label $@L_i$ called accepting activity label.
2. If LTS $L=L_1||\ldots||L_k$ and there are more than one subcomponents have accepting states, then regard their accepting activity labels as same , that is $@L_i=@L_j$, $i\neq j$.
3. The executions of accepting activities comply with parallel rule (3) in section 3.

When service L_W goes into accepting state $s=(s_1,\ldots,s_k)$, all of its subservices go into their accepting states and they will execute the same accepting activity labeled with $@L_W$. Thus a normal finite execution with accepting state will become infinite execution with postfix of $@L_W$ s $@L_W$ s…$@L_W$…., where s is a accepting state. If L_W goes into a deadlock state s, there is at least one state s_i which L_i is not terminated and it is waiting for another subservice synchronization step. So an execution with final deadlock state of L_W will be finite execution. Figure 3.1(3),(4) is an extended LTS.

The real deadlock checking of extended LTS L_W may be completed with DFS algorithm through the whole graph space of L_W by judging whether its state s has successive state.

Let $L_W=L_1||\ldots||L_k$ represents compositional web service. LTS L_M represents the activity chain mode LTS. $L=L_W||L_M$. We give preconditions of no deadlock state for compositional web service compliance verification below.

Precondition1: L_i has no deadlock state.

Precondition 2: L_W has no deadlock state.

Precondition3: L has no deadlock state.

The preconditions above hold under the deadlock checking described above and deadlock repairing.

B. The Compliance Verification forActivity Chain Precondition Mode

In this section, we first give the definition of compliance verification and then the characteristics about activity chain modes. Finally, check the compliance of web service for activity chain modes with reachable analysis of LTS.

Definition4.2. let $L_M=<S_M,A_M,\rightarrow_M,s_{M0}>$ is a LTS of activity chain in figure 3.1. $A_M=A_{M1}\cup A_{M2}\cup\{@L\}$ or $A_M=A_{M1}\cup A_{M2}$, where $A_{M1}\subseteq$ Act is normal activity labels, $A_{M2}=\{!a|a\in A_{M1}\}$, @L is an accepting activity label defined in 4.1.If $a\in A_M$, $b\in$Act, then b is called matching with a, iff $a\in A_{M1}\wedge b=a$ or $a\in A_{M2}\wedge a=!c\wedge b\neq c$. denote it b~a. If b is an accepting activity label @L', then b is called matching with a, iff $a\in A_{M2}$ or a=@L. denote it b~a.

If $b\in$ Act $\cup\{@L\}$ and $\exists a\in A_M$ satisfy that b~a, then denote $b\in A_M$.

Definition4.3. let $L_W=<S_W,A_W,\rightarrow_W,s_{W0}>$,$L_M=<S_M,A_M,\rightarrow_M,s_{M0}>$ is LTS of compositional web service and activity chain mode respectively. ω is an infinite or finite activity chain of L_W. The projection $\downarrow_M$: $A_W^*\rightarrow A_M^*$ is defined inductively below.

(1) $(.)\downarrow_M=(\ )$.

(2) If $a\in A_M$, then $((a)\frown\sigma)\downarrow_M=(a)\frown(\sigma\downarrow_M)$
(3) If $a\notin A_M$, then $((a)\frown\sigma)\downarrow_M=\sigma\downarrow_M$

Where, $\frown$ is a connector between two symbols. In fact, $\downarrow_M$ is a filter function for traces of service that only reserve the symbols in A_M and discard the symbols not in A_M.

Definition4.4. (compliance for activity chain mode) let $L_W=L_1||L_2||\ldots||L_k$ is a compositional web service. L_M is an activity chain mode defined in 2.2. L_W and L_M satisfy preconditions in section 4.1. If L_M and $\sigma\downarrow_M$ is a trace of L_M for any trace of L_W, σ, then call compositional web service L_W is compliant with activity chain mode L_M and denote it $L_W\models L_M$.

Lemma4.1. let σ is a trace of $L_W=L_1||L_2||\ldots||L_k$. $L_M=L_i$. then $\sigma\downarrow_i$ must be a trace of L_i.

Proof: let $\sigma=<a_1a_2\ldots>$,$\sigma\downarrow_i=<a_{i1}a_{i2}\ldots>$. From the definition4.3, $a_{ik}\in A_i$,k=1,….so $\sigma\downarrow_i$ is a activity chain of L_i. the activity labels $a_1,a_2,\ldots a_{ik-1}$ occurred before a_{i1} are not in A_i. So a_{i1} is first activity label in σ of L that is also in A_i. a_{i1} is first activity label in L_i from initial state related to the σ. Similarity σ has no any other activity label in A_i between a_{ij} and a_{ij+1}. So $\sigma\downarrow_i$ is a L_i trace. □

Lemma4.2. assume that L_{C_PR} is an activity chain precondition mode. If no considering the repetition of accepting activity label $@L_{C\_PR}$, the normal conclusion execution trace of L_{C_PR} will be the form of$(!a)^*a(!a_1)^*a_1(!a_2)^*a_2\ldots(!a_n)^*a_n$.

The proof of theorem is obvious. In figure 3.1 (3), for every normal conclusion execution of L_{C_PR} ,ρ, ρ must start from initial state and go into accepting state s_n and followed by infinite accepting activities @ L_{C_PR}. So the trace generated by ρ must be $(!a)^*a(!a_1)^*a_1(!a_2)^*a_2\ldots(!a_n)^*a_n\ (@L_{C\_PR})^\omega$. The lemma holds without considering the infinite accepting activities postfix $(@L_{C\_PR})^\omega$.

It is clear that every normal conclusion execution of L_{C_PR}, ρ, must contains all activities occurred in the order of that in C and activity a. Activity a is the first activity apart from initial state and finally the execution must contain activities in C. So the chain precondition mode a ***C_PR*** C holds. Conversely, if a behavior specification is expressed as a ***C_PR*** C, C and a must occur in any execution of compositional web service,ρ, then ρ must generate traces of the sub form $(!a)^*a(!a_1)^*a_1(!a_2)^*a_2\ldots(!a_n)^*a_n$,. Since the parallel operation rule in definition3.2, the LTS of the specification must also be such form. So L_{C_PR} reflects the exact operation semantic of activity chain precondition mode.

Theorem 4.3.Assume that $L_{W=}<S_W, A_W,\rightarrow_W,s_{W0}>$ is a compositional web service. $C=<a_1, a_2,\ldots a_n>$ is an activity chain. $L_{C\_PR}=<S_{C\_PR}, A_{C\_PR}, \rightarrow_{C\_PR}, s_{C\_PR0}>$ is an activity chain precondition mode LTS. $L=L_W||L_{C\_PR}=(S,A,\rightarrow,s_0)$. Then compositional web service L_W is compliant with a ***C_PR*** C iff any ring in L is initial reachable and must contains the transition labeled with activity $@L_{C\_PR}$.

Proof: for sufficient case, let ρ is any execution of L. ρ must be infinite execution because of preconditions. ρ must have a ring for only finite states. The transition labeled with the activity $@L_{C\_PR}$ must occur in ρ due to sufficient condition. Let σ is a trace corresponding to ρ.So $@L_{C\_PR}$ must occur in σ. Let $\sigma\downarrow_{C\_PR}$ is a projection on A_{C_PR}. $\sigma\downarrow_{C\_PR}$ must be a trace of L_{C_PR} related to ρ for the lemma 4.1. Thus $@L_{C\_PR}$ must occur in $\sigma\downarrow_{C\_PR}$. The accepting state s_n of L_{C_PR} must reach through the execution of ρ.then conclude that $\sigma\downarrow_{C\_PR}$ must be the form of $(!a)^*a\ (!a_1)^*a_1(!a_2)^*a_2\ \ldots(!a_n)^*a_n$ and σ must contain activity chain C and a. meanwhile a must occur before C. Because ρ is any execution of L. So a ***C_PR*** C holds.

For the necessary case, if there is a reachable ring that contains no $@L_{C\_PR}$. If (s_W, s_n) occur in this ring which s_n is accepting state of L_{C_PR}. Then s_n has only one successor activity $@L_{C\_PR}$ and $@L_{C\_PR}$ must not occur for the assumption. So it is only the case that the direct successor activity of (s_W, s_n) will be $a\in A\backslash A_{C\_PR}$. The compositional web service L_W has the infinite execution disjoint with L_{C_PR} and s_n must not occur in this execution. It is a contradiction. So (s_W, s_n) does not occur in ring and the activity chain Q does not exist. □

An algorithm for checking the compliance of compositional web service can be obtained from the theorem4.3 listed below.

Check_C_PR_Chain(LTS L_i, activity chain Q)

(1) Constructing activity chain mode $L_{C\text{-}PR}$ for Q according the rule1 in figure 3.1.
(2) Extending $L_{C\text{-}PR}$ and L_i according to the rules in section IV A..
(3) $L=L_1||\ldots||L_m||\ L_{C\text{-}PR}$. DFS search throughout L, pick out every ring in L if possible or if no any ring in L goes to (5).
(4) For every ring picked out, checking whether $@L_{C\text{-}PR}$ is in the ring or not. If not, then L_W is incompliant with a ***C-PR*** C and exit. If yes, goes to (3).
(5) If there is a ring in L, then L_W is compliant with a ***C-PR*** C, otherwise, L_W is incompliant with a ***C-PR*** C.

C. Compliance Verifications for Chain Response Mode

Chain Response mode requires that when activity chain C occurs in certain trace of L_W, σ, it must lead to activity a occur in the certain time of future. Their corresponding activity chain mode LTS may be extended by the rule in section IV A. Figure 3.1.(4) L_{C_RE} is its corresponding LTS extended. L_{C_RE} are similar with chain precondition mode L_{C_PR} in structure. The compliance verifications is similar with the compliance verification for chain precondition mode.

Theorem4.4.Assume that $L_{W=}<S_W,A_W,\rightarrow_W,s_{W0}>$ is a compositional web service. $C=<a_1,a_2,\ldots a_n>$ is an activity chain and a is an activity. $L_{C\_RE}=<S_{C\_RE},A_{C\_RE},\rightarrow_{C\_RE},s_{C\_RE0}>$ is an LTS of activity chain precondition mode. $L=L_W||L_{C\text{-}RE}=(S,A,\rightarrow,s_0)$. Then compositional web service L_W is compliant with a ***C_RE*** C iff any ring in L is initial reachable and must contains the transition labeled with activity $@L_{C\text{-}RE}$.

V EXAMPLE ANALYSIS

In this section, we analysis the example presented in section II and check its compliance for Binary activity chain modes. In Figure 5.1, we give the LTS description of F_H and activity chain mode LTS. For the sake of simplicity, we omit other LTSs and its parallel of F and H. Because F and H only interact with F_H in the orchestration service and the client also interacts with F_H only. The activities in modes are activities of L_{F_H} which are focused by Client. So we simply use L_{F_H} instead of the parallel with L_F and L_H. It can be seen that there is a ring of $L_{F\text{-}H}||L_{C\text{-}PR}$ that the activities labeled are not matching with the accepting activity $@L=@L_{C\text{-}PR}=@L_{F\text{-}H.}$ In fact, the ring is (13, 2)e_2 (15,2) e*(13,2). So the service is not compliant with activity chain precondition mode. Similarity, L_W is not compliant with the activity chain mode a ***RE*** C.

VI CONCLUSION

In this paper, we propose the concept of behavior specification based on activity chain. Its basic element is activity chain and its granularity between

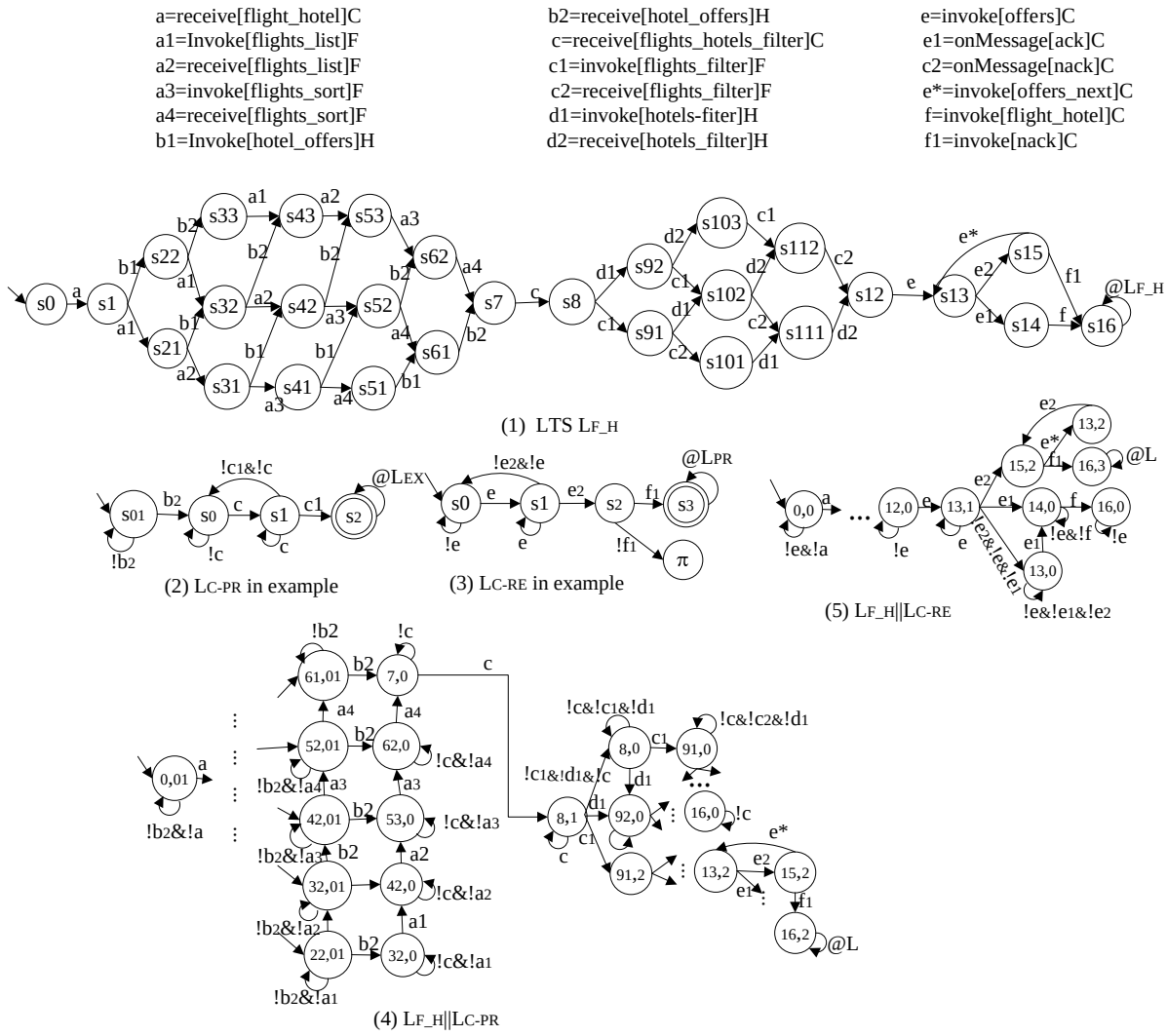

Figure 5.1. The LTSs of compositional web service F_H and modes and paralle

activity and scenario. Referencing the idea of attribute patterns based on activity, we propose two binary behavior modes based on activity chain that are precondition mode, response mode. These modes can be used to describe behavior requirements for compositional web services. They are suitable for many practical cases. The scope in attribute pattern can not adopted in activity chain modes. The reason is that the scope can also be regarded as the result of "and" operation of multi modes, for example, the scope "before" can be seen as a precondition mode. Without the scope, activity chain modes can be more simple and concise. In order to verify the compliance for above modes, we extend LTS model to exclude deadlock state. Encode modes into LTS. Also by translating BPEL process into LTS, we give out the sufficient and necessary condition to check the compliance of compositional web service for modes. Finally, an example and its analysis are illustrated in

paper. Future work may be consideration of optimization of verification.

REFERENCES

[1] M.Tarek, C. Boutrous-Saab. "Verifying correctness of Web services choreography," Proceedings of the European Conference on Web Services, 2006, pp.306-318.

[2] R.Kazhamiakin, M. Pistore. A parametric communication model for the verification of BPEL4WS Compositions. Lecture Notes in Computer Science, 2005, v3670, pp.318-332.

[3] L.Bordeaux, G.Salaun, D.Berardi,M. Mecella. When are two Web services compatible? Lecture Notes in Computer Science, 2005, v3324, pp.15-28.

[4] C.Ouyanga,_E. Verbeekb, V.D. Aalsta. Formal semantics and analysis of control flow in WS-BPEL. Science of Computer Programming, , 67(4),pp.162–198, 2007.

[5] S.Nakajima. Model-checking behavioral specification of BPEL application. Electronic Notes in Theoretical Computer Science, 151(2),pp.89-105, 2006.

[6] Z.Qiu, S.Wang, G.Pu, X.Zhao. Semantics of BPEL4WS-like fault and compensation handling. Lecture Notes in Computer Science, v3582, pp.350-365, 2005.

[7] X.Fu, T. Bultan, J.Su. Analysis of interacting BPEL Web services. Proceedings of the 13th International Conference on World Wide Web, 2004, pp. 624-630.

[8] A.B.Can, T.Bultan, X.Fu. Design for verification for asynchronously communicating Web services. Proceedings of the 14th International Conference on World Wide Web, 2005, pp.750-759.

[9] M. Mongiello, D.Castelluccia. Modeling and verification of BPEL business processes. Proceedings of the 4th Workshop on Model-Based Development of Computer-Based Systems.2006, pp.144-148

[10] H.Foster, S.Uchitel, J.Magee, J. Kramer. Model-based verification of Web service compositions. IEEE International Conference on Automated Software Engineering, 2003, pp.152-163.

[11] V.D. Aalst. Conformance checking of service behavior. ACM Transactions on Internet Technology, 8(3):1-13, 2008.

[12] M. Pistore, M. Roveri, P.Busetta. Requirements-driven verification of Web services. Electronic Notes in Theoretical Computer Science, 105(3), pp. 95–108,2004.

[13] M.Rouached, C. Godart. Requirements-driven verification of WSBPEL processes. IEEE International Conference on Web Services, 2007, pp.354–363.

[14] Hu jun, Yu xiao-feng, Zhang yuan, etc.. checking Component-based Design for Scenario-baesd specifica- tions. Chinese Journal of Computer. 29(4), pp.513-525,2006.

[15] M.B. Dwyer, G.S. Avrunin, J.C. Corbett. Patterns in property specifications for finite-state verification. Proceedings of the 1999 International Conference on Software Engineering, 1999, pp.411-420.

[16] J. Yu, T.P. Manh, J. Han, U. Jin, Y. Han, J.W. Wang. Pattern based property specification and verification for service composition. Lecture Notes in Computer Science, 4255, pp.156-168, 2006.

[17] S.C. Cheung J. Kramer. Checking safety properties using compositional reachability analysis. ACM Transactions on Software Engineering and Methodology, 8(1),pp. 49-78.,1999.

[18] D. Giannakopoulou. Model checking for concurrent software architectures. Imperial College of Science, Technology and Medicine University of London. Ph.D.thesis, 1999.

Bo Chen, born on Dec.1963, graduated from Chinese Science Institute for M.S. degree in the field of control theory in 1990 and now is a Ph.D. candidate of Tongji University in the field of software theory. His major study interests include trusted software and model checking.

He has been for teaching and researching work for about twenty years in Guangxi Unoversity of Technology, Liuzhou, China. As an assistant professor, he has lectured many courses in computer science such as software engineering, database system, etc. He has published about twenty research papers in academic journal and international conference such as Computer Science, Geomatics and Information Science of Wuhan University, Chinese Journal of Computer, Niss 2008, APCIP 2009, etc.

Chungui Li, born on Aug. 1968, graduated from Beijing Institute of Technology for Ph.D. in artificial intelligence science in 2003. His main research interests include machine learning, intelligent systems, information retrieval and data mining.

He has been teaching and researching in Guangxi University of Technology, Liuzhou, China for many years. As an associate professor, he has published many papers in national academic Journal and international conference.

Qixian Cai, born on Sept. 1948, as a graduate student, graduated from SouthEast University electronics in 1986, Nanjing, China. His major fields of study include temporal database technique, computer architecture, and computer based education.

He was engaged in higher education work for 30 years，and once served as a vice-director of Computer engineering department, vice-director of Academic administration etc. Now, he is professor of Dept. of computer engineering of GuangXi University of Technology, in LiuZhou city of GuangXi province of China. He published 12 books and 3 softwares. Was a first author totaled to publish more than 50 papers in academic journals. The major publishing books and papers have: Computer Architecture, Beijing, China: Electronics Industrial Publisher; Computer Application in Education:Principle and Practice, GuiLin, China: GuangXi Normal University publisher; C Programming and Application, Beijing, China: Electronics Industrial Publisher; Research into C-Temporal Relation Data Model, Journal of Natural Science of Hunan Normal University, Changsha, China, 2004, vol.27, no.1, pp.18-22; Researching into the Method of Key-Time based on the C-TRDM and the Time Filtration Operator, Journal of natural science of Hunan Normal University, Changsha, China, Vol.28 No.3 Mar., 2004. pp13-17.

APPENDIX A: BPEL PROCESS OF COMPOSITIONAL WEB SERVICE F_H

```
<? xml version="1.0" encoding="UTF-8"?>
<process name=”F_H”
targetNamespace=”http://F_Htravel.com/bpel/travel/”
 xmlns:tns=”http://f_htravel.com/bpel/travel/”
 xmlns=”http://schemas.xmlsoap.org/ws/2004/3/03/businessprocess/”
 xmlns:trv=”http://f_htravel.com/bpel/travel/”
 xmlns:fli=”http://flighttravel.com/bpel/flight/”
 xmlns:hot=”http://hoteltravel.com/bpel/hotel/”>
 <partnerlinks>
   <partnerlink name=”Travel” partnerLinkType=”trv: TravelLT”
     myRole=”F_HService” partnerRole=”TravelCustomer”/>
 <partnerlink name=”Travelfilter” partnerLinkType=”trv:
     TravelFilterLT” myRole=”F_HFilterService”
     partnerRole=”TravelFilterCustomer”/>
```

```
    <partnerlink name=”FlightList" partnerLinkType=”fli:
      FlightListLT” myRole=”FlightListRequester”
      partnerRole=”FlightListProvider”/>
    <partnerlink name=”FlightSort” partnerlinkType=”fli:
      FlightSortLT” myRole=”FlightSortRequester”
      partnerRole=”FlightSortProvider”/>
  <partnerlink name=”FlightFilter” partnerlinkType=”fli:
      FlightFilterLT” myRole=” FlightFilterRequester”
      partnerRole=”FlightFilterProvider”/>
    <partnerlink name="HotelOffer" partnerLinkType=”hot: HotelLT”
      myRole=”HotelOfferRequester” partnerRole="HotelProvider"/>
    <partnerlink name=”HotelFilter” partnerLinkType=”hot:
      HotelFilterLT” myRole=”HotelFilterRequester”
      partnerRole=”HotelFilterProvider”/>
  <partnerlink name=”ack” partnerLinkType=”trv: ackLT”
      myRole=”ackservice””/>
  <partnerlink name=”nack” partnerLinkType=”trv: nackLT”
      myRole=”nackservice””/>
   </partnerlinks>
  <variables>…</variables>
  <sequence>
<receive name=”X” partnerlink=”HotelLT” portType=”hot:
              HotelCallbackPT” operation=”hotel_offersCallback”
              sourcelinkname=”XtoY” Tc=”getLinkStatus ('XtoY')”/>
      </sequence>
      </flow>
        <receive name=”Y” partnerlink=”TravelFilter” portType=”trv:
          TravelFilterPT” operation=”flights_hotels_filter”
          targetlinkname=”XtoY”/>
      <flow>
       <sequence>
          <invoke partnerlink=”FlightFilterLT” portType=”fli: FlilgtPT
            operation=”flights_filter”/>
          <receive partnerlink=”FlightFilterLT” portType=”fli:
            FlilgtCallbackPT” operation=”flights_filtCallback”/>
       </sequence>
       <sequence>
```

```
<receive partnerlink=”Travel” portType=”trv: TravelPT”
  operation=”flight_hotel” createInstance=”yes”/>
  <flow superessJoinFailure=”yes”>
  <links>
        <link name=”XtoY”/>
  </links>
   <sequence>
      <invoke partnerlink=”FlightList” portType=”fli: FlightPT”
        operation=”fligh_list”/>
      <receive partnerlink=”FlightList” portType=”fli:
        FlightCallbackPT” operation=”flight_listCallback”/>
    <invoke partnerlink=”FlightSort” portType=”fli: FlightPT”
      operation=”fligh_sort”/>
   <receive partnerlink=”FlightSort” portType=”fli:
        FlightCallbackPT” operation=”flight_sortCallback”/>
   </sequence>
   <sequence>
      <invoke partnerlink=”HotelLT” portType=”hot: HotelPT”
        operation=”hotel_offers”/>

  <invoke partnerlink=”HotelFilterLT” portType=”hot: HotelPT”
    operation=”hotels_filter”/>
  <receive partnerlink=”HotelFilterLT” portType=”hot:
      HotelFilterCallbackPT” operation=”hotels_filterCallback”/>
 </sequence>
</flow>
  <invoke partnerlink=”TravelFilter” portType=”trv:
   TravelFilterCallbackPT” operation=”offersCallback”>
<pick createInstance="no">
  <onMessage partnerlink=”ackLT” portType=”trv: ackPT”
   operation=”flight_hotelCallback”> </onMessage>
<onMessage partnerlink=”nackLT” portType=’trv: nackPT”
  operation=”offernextCallback”> </onMessage>
</pick>
</sequence>
</process>
```