Journal of Southeast University (English Edition) Vol. 24, No. 3, pp. 300 – 307 Sept. 2008 ISSN 1003—7985

Typed formal model for WS-CDL specification of web services composition

Gu Xiwu Li Ruixuan Lu Zhengding

(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: In order to formally reason and verify web services composition described by web services choreography specification WS-CDL, a typed formal model named typed abstract WS-CDL (web services choreography description language) for WS-CDL specifications is proposed. In typed abstract WS-CDL, the syntax of type and session, typing rules and operational semantics are formalized; the collaborations of web services are formally described by sessions; the operational semantics of a session can help to formally reason the execution of the choreography; the typing rules can help to formally check the data type consistency of exchanged information between web services and capture run-time errors due to type mismatches. Particularly, the concepts of type assumption set extension and type assumption set compatibility are proposed, and the merging algorithm of type assumption sets is defined so as to eliminate type assumption conflict. Based on the formal model, typed mapping rules for mapping web services choreography to orchestration is also defined. With the typed mapping rules, orchestration stubs and their type assumption sets can be generated from a given choreography; thus, web services composition can be verified at choreography and orchestration levels, respectively. The model is proved to have properties of type safety, and how the model can help to reason and verify web services composition is illustrated through a case study.

Key words: typed model; web services composition; web services choreography description language

Nowadays, standards for web services composition cover two different points of view: choreography and orchestration[1]. Choreography describes the interactions between web services from a global perspective while orchestration describes the interactions in which a given service can engage with other services. The web services choreography description language (WS-CDL)[2] is the latest and most important choreography specification. However, WS-CDL is an XML-based descriptive language, lacking formal models to express the semantics of WS-CDL accurately and formal verification mechanisms to ensure the correctness of a composite web service such as behavior compatibility and data type consistency.

In this paper, we propose a typed formal model named typed abstract WS-CDL for WS-CDL specifications. The model can be used to reason on the execution of web service choreography and check the type consistency of exchanged information. The typed abstract WS-CDL can be proved to have properties of type safety. Moreover, we define a set of typed mapping rules for generation orchestration stubs described by a typed pi-calculus process from a given web service choreography.

1 Related Work

There are some existing works on formally modeling and verifying web services composition. Salaun et al.[3] formally described and reasoned web services composition from choreography and orchestration views based on process algebra CCS; Brogi et al.[4] proposed a CCS-based formal model for web service choreography interfaces (i.e., WSCI); Busi et al.[5] proposed a formal framework of WS-CDL; Yeung et al.[6] formally verified web services composition described by WS-CDL based on process algebra CSP; Zhao et al.[7] proposed a formal model CDL for WS-CDL specifications.

In addition to the works listed above, some proposed formal models based on type theory can be used to formally check data type consistency of exchanged information between web services. Gay and Hole[8] proposed a typed formal model based on session types which can be used to verify the dynamic behavior and type consistency of web services; Pahl[9] proposed a pi-calculus-based formal framework for the composition of components in which the interactions between components can be described by port type and contract.

2 Typed Abstract WS-CDL

2.1 Notation definitions for background concepts

Definition 1 (session) A session, denoted by S, represents an activity that may be performed by one or more of the participants of composite web services.

Definition 2 (role) A role, denoted by r, represents a participant of a session. The set of roles that represents the participants of a session S is denoted as $S[R]$. The set of variables and operations of role r are denoted as $r[V]$ and $r[O]$, respectively. Moreover, we denote a variable x of role r as $r[x]$ and an operation o of role r as $r.o$.

Received 2008-04-15.

Biographies: Gu Xiwu (1967—), male, doctor, lecturer, guxw_wang@sina.com; Li Ruixuan (1974—), male, doctor, associate professor, rxli@hust.edu.cn.

Foundation items: The National Natural Science Foundation of China (No. 60403027, 60773191, 70771043), the National High Technology Research and Development Program of China (863 Program) (No. 2007AA01Z403).

Citation: Gu Xiwu, Li Ruixuan, Lu Zhengding. Typed formal model for WS-CDL specification of web services composition[J]. Journal of Southeast University (English Edition), 2008, 24(3): 300 – 307.

Definition 3 (channel) A channel is a point where an interaction between participants occurs. A channel ch is located at the role of information receiver, which is denoted as ch@ $r.\ o$. The set of channels through which a session is performed is denoted as $S[C]$.

Definition 4 (proposition) A proposition, denoted by p, represents the guard or repeat condition of a workunit activity in WS-CDL. We denote the set of roles involved in a proposition p by $p[R]$, since the evaluation of p may be associated with variables of more than one role due to the "global" characteristics of WS-CDL specifications.

2.2 Syntax

2.2.1 Syntax of type

Types	$S, T ::= B \mid \text{VS}, \text{VT}$
	$B ::= \text{bool} \mid \text{int} \mid \text{real} \mid \ldots$ (basic data type)
	$\text{VS}, \text{VT} ::= R(B) \mid W(B) \mid \text{RW}(B)$ (variable type)
	$\text{C} ::= \uparrow r.\ o(B) \mid \downarrow r.\ o(B) \mid \updownarrow r.\ o(B)$ (channel type)
Variables	$V ::= r[x]$
Channel	$\text{CH} ::= \text{ch}$
Type assumption set	$\Gamma ::= \Phi_\Gamma \mid \Gamma, r[x]: \text{VT} \mid \Gamma, \text{ch}: C$
Subtype relation	$\leqslant_T ::= \Phi_\leqslant \mid \leqslant_T, S \leqslant T$

The letters S, T range over basic data types and variable types. Basic data types ranged over by B stand for arbitrary basic data types. The variable types ranged over by VS, VT include three types of variables denoted by type construct R, W and RW: $R(B)$ is the type of variables that can be read and the value read from is assumed to have type B; $W(B)$ represents the type of variables that can be written, and the values written to have type B; $\text{RW}(B)$ represents the type of variables that can be read and written, the value read and written has the same type B.

The channel types ranged over by C include three types of channels: $\uparrow r.\ o(B)$ stands for a request channel type; the target of information exchange is operation o of role r, and the type of information passed through is a type or subtype of B; similarly, $\downarrow r.\ o(B)$ stands for a response channel type; $\updownarrow r.\ o(B)$ stands for a request-response channel type. The arbitrary variable of role r ranged over by $r[x]$ and channel is ranged over by ch.

The type assumption set $\Gamma = \{a_i: T_i \mid i \neq j \Rightarrow a_i \neq a_j\}$, where $a_i: T_i$ is a type assumption that name a_i has type T_i. Let $\text{Dom}(\Gamma) = \{a \mid a: T \in \Gamma\}$, then we have $\forall a \in \text{Dom}(\Gamma), \Gamma(a) = T$. An empty type assumption set is denoted as Φ_Γ. The subtype relation on type set T is denoted by $\leqslant_T$, $\leqslant_T = \{T_i \leqslant T_j \mid T_i, T_j \in T \text{ and } i \neq j \Rightarrow T_i \neq T_j\}$ where $T_i \leqslant T_j$ denotes that T_i is a subtype of T_j. An empty subtype relation is denoted as $\Phi_\leqslant$. Name a can be proved to have type T under Γ and $\leqslant_T$ is denoted as $\Gamma, \leqslant_T \vdash a: T$. Contrarily, $\Gamma, \leqslant_T \overline{\vdash} a: T$ asserts that the type of name a does not have type T under Γ and $\leqslant_T$. Type S can be proved to be a subtype of type T under the subtype relation, $\leqslant_T$ is denoted as $\leqslant_T \vdash S \leqslant T$. The session S is called a well typed session, denoted by $\Gamma, \leqslant_T \vdash S$, if it can be proved that session S does not have any error of type mismatch under Γ and $\leqslant_T$. Contrarily, $\Gamma, \leqslant_T \overline{\vdash} S$ denotes that S is not a well typed session.

2.2.2 Syntax of session

$$S ::= S_{\text{atom}} \mid S_1 + S_2 \mid S_1 \| S_2 \mid S_1.\ S_2 \mid [P_{\text{g}}][P_{\text{rep}}] * S \mid \text{Err}$$
$$S_{\text{atom}} ::= S_{\text{no}} \mid S_{\text{silent}} \mid S_{\text{assign}} \mid S_{\text{req}} \mid S_{\text{resp}} \mid S_{\text{req-resp}} \mid \text{NULL}$$
$$S_{\text{no}} ::= r(0)\ S_{\text{silent}} ::= r(\tau)\ S_{\text{assign}} ::= \text{assign}(r[x] \leftarrow r[e])$$
$$S_{\text{req}} ::= \text{req}(r_1, r_2, \text{ch@}\ r_2.\ o, r_1[x] \rightarrow r_2[y])\ S_{\text{resp}} ::= \text{resp}(r_1, r_2, \text{ch@}\ r_1.\ o, r_1[x] \leftarrow r_2[y])$$
$$S_{\text{req-resp}} ::= \text{req-resp}(r_1, r_2, \text{ch@}\ r_2.\ o, r_1[x] \rightarrow r_2[y], r_1[s] \leftarrow r_2[t])$$

In the definitions above, a sessions S can be a non-deterministic choice between sessions S_1 and S_2 ($S_1 + S_2$), a parallel execution of sessions S_1 and S_2 ($S_1 \| S_2$), a sequential execution of sessions S_1 and S_2 ($S_1.\ S_2$), a conditional iterative execution of an enclosed session ($[P_{\text{g}}][P_{\text{rep}}] * S$), a session in which a run-time error has occurred (Err) and an atomic session (S_{atom}).

The atomic session S_{no} denotes that role r does not perform any action; S_{silent} denotes that role r performs an internal silent action τ; S_{assign} denotes that a value assignment from variable e to variable x on role r; S_{req} represents a request interaction from role r_1 to role r_2 through channel ch@ $r_2.\ o$, in which the request message is sent from $r_1[x]$ to $r_2[y]$. Similarly, S_{resp} represents a response interaction from role r_2 to role r_1; $S_{\text{req-resp}}$ represents a request-response interaction between role r_1 and role r_2.

2.3 Typing rules

1) Variable and channel typing rules

$$\text{TVAR: } \frac{\Gamma(r[x]) = VT}{\Gamma, \leqslant_T \vdash r[x]: \text{VT}}, \quad \text{TCH: } \frac{\Gamma(\text{ch}) = C}{\Gamma, \leqslant_T \vdash \text{ch}: C}, \quad \text{TSUB: } \frac{\Gamma, \leqslant_T \vdash V: \text{VS} \wedge \leqslant_T \vdash \text{VS} \leqslant \text{VT}}{\Gamma, \leqslant_T \vdash V: \text{VT}}$$

2) Subtyping rules

$$\text{TSELF: } \frac{}{\leqslant_T \vdash T \leqslant T}, \quad \text{TTRANS: } \frac{\leqslant_T \vdash T_1 \leqslant T_2 \wedge \leqslant_T \vdash T_2 \leqslant T_3}{\leqslant_T \vdash T_1 \leqslant T_3}$$

$$\text{TRSR: } \frac{\leqslant_T \vdash B_1 \leqslant B_2}{\leqslant_T \vdash R(B_1) \leqslant R(B_2)}, \quad \text{TRWRW: } \frac{\leqslant_T \vdash B_1 \leqslant B_2 \ , \quad \leqslant_T \vdash B_2 \leqslant B_1}{\leqslant_T \vdash \mathrm{RW}(B_1) \leqslant \mathrm{RW}(B_2)}$$

$$\text{TRWSR: } \frac{}{\leqslant_T \vdash \mathrm{RW}(B) \leqslant R(B)}, \quad \text{TRWSW: } \frac{}{\leqslant_T \vdash \mathrm{RW}(B) \leqslant W(B)}, \quad \text{TWSW: } \frac{\leqslant_T \vdash B_1 \leqslant B_2}{\leqslant_T \vdash W(B_2) \leqslant W(B_1)}$$

3) Session typing rules

$$\text{TNULL: } \frac{}{\Gamma, \leqslant_T \vdash \mathrm{NULL}}, \quad \text{TNO: } \frac{}{\Gamma \ , \leqslant_T \vdash r(0)}, \quad \text{TTAU: } \frac{}{\Gamma, \leqslant_T \vdash r(\tau)}$$

$$\text{TREQ: } \frac{\Gamma \vdash \mathrm{ch}: \uparrow r_2.\, o(B) \wedge \Gamma, \leqslant_T \vdash r_1[x]: R(B_1) \wedge \Gamma \ , \leqslant_T \vdash r_2[y]: W(B_2) \wedge \leqslant_T \vdash B_1 \leqslant B \leqslant B_2}{\Gamma \ , \leqslant_T \vdash S_{\mathrm{req}}}$$

$$\text{TRESP: } \frac{\Gamma \vdash \mathrm{ch}: \downarrow r_1.\, o(B) \wedge \Gamma, \leqslant_T \vdash r_1[x]: W(B_1) \wedge \Gamma, \leqslant_T \vdash r_2[y]: R(B_2) \wedge \leqslant_T \vdash B_2 \leqslant B \leqslant B_1}{\Gamma, \leqslant_T \vdash S_{\mathrm{resp}}}$$

$$\text{TRR: } \frac{\begin{array}{c}\Gamma \vdash \mathrm{ch}: \updownarrow r_2.\, o(B) \wedge \Gamma, \leqslant_T \vdash r_1[x]: R(B_1) \wedge \Gamma, \leqslant_T \vdash r_2[y]: W(B_2) \wedge \leqslant_T \vdash B_1 \leqslant B \leqslant B_2 \wedge \Gamma, \leqslant_T \\ \vdash r_1[s]: W(B_4) \wedge \Gamma, \leqslant_T \vdash r_2[t]: R(B_3) \wedge \leqslant_T \vdash B_3 \leqslant B \leqslant B_4\end{array}}{\Gamma, \leqslant_T \vdash S_{\mathrm{req\text{-}resp}}}$$

$$\text{TASSIGN: } \frac{\Gamma, \leqslant_T \vdash r[x]: W(B_2) \wedge \Gamma, \leqslant_T \vdash r[e]: R(B_1) \wedge \leqslant_T \vdash B_1 \leqslant B_2}{\Gamma, \leqslant_T \vdash S_{\mathrm{assign}}}, \quad \text{TITR: } \frac{\Gamma, \leqslant_T \vdash S}{\Gamma, \leqslant_T \vdash [P_{\mathrm{repeat}}] * S}$$

$$\text{TWORKUNIT: } \frac{\Gamma, \leqslant_T \vdash S}{\Gamma, \leqslant_T \vdash [P_{\mathrm{g}}][P_{\mathrm{repeat}}] * S}, \quad \text{TSEQ: } \frac{\Gamma, \leqslant_T \vdash S_1 \wedge \Gamma, \leqslant_T \vdash S_2}{\Gamma, \leqslant_T \vdash S_1.\, S_2}$$

$$\text{TCHOICE: } \frac{\Gamma, \leqslant_T \vdash S_1 \wedge \Gamma, \leqslant_T \vdash S_2}{\Gamma, \leqslant_T \vdash S_1 + S_2}, \quad \text{TPAR: } \frac{\Gamma, \leqslant_T \vdash S_1 \wedge \Gamma, \leqslant_T \vdash S_2}{\Gamma, \leqslant_T \vdash S_1 \,\|\, S_2}$$

In the typing rules, above the line are premises and below the line are conclusions. "Err" is not well-typed in any case. Rules TVAR, TCH, TSUB, TSELF, TTRANS, TRWSR, TRWSW, TRSR, TWSW, TRWRW are standard rules in type theory.

Rules TNULL, TNO and TTAU show that atomic sessions NULL, S_{no} and S_{silent} are well-typed sessions in any case; rule TREQ shows that an atomic session req $(r_1, r_2, \mathrm{ch}@r_2.\ o, r_1[x] \rightarrow r_2[y])$ is a well typed session if the type of information passed through ch is B; the target of information is operation o of role r_2; the type of variable $r_1[x]$ on sender is $R(B_1)$; the type of variable $r_2[y]$ on receive r is $W(B_2)$, and $B_1 \leqslant B \leqslant B_2$. The case for atomic sessions $S_{\mathrm{resp}}, S_{\mathrm{req\text{-}resp}}, S_{\mathrm{assign}}$ are similar to S_{req}, which are denoted by rule TRESP, TRR, TASSIGN, respectively. The rules TWORKUNIT and TITR show that if the enclosed session S is well-typed, then $[P_{\mathrm{g}}][P_{\mathrm{rep}}] * S$ and $[P_{\mathrm{rep}}] * S$ are well-typed. The rules TSEQ, TCHOICE and TPAR show that the sequence, choice and parallel executions of two sessions are well-typed if each is well typed.

2.4 Operational semantics

The operational semantic of typed abstract WS-CDL is given through a labelled transition system which describes the dynamic evolution of a session. The transitions are of kind $S \xrightarrow{p,\alpha} S'$ for atomic actions ranged over by α, meaning that if proposition p holds, then session S has an atomic action α leading to session S'. The atomic action α is defined as

$$\alpha ::= S_{\mathrm{no}} \mid S_{\mathrm{silent}} \mid S_{\mathrm{assign}} \mid S_{\mathrm{req}} \mid S_{\mathrm{resp}} \mid S_{\mathrm{req\text{-}resp}} \mid \mathrm{NULL}$$

Rule NATOM states that the execution of an atomic session results in a NULL session; NSEQ1 states that a sequentially composite session $\alpha.\, S$ evolves into S after a one-step execution of α; rules NSTRU, NSEQ2, NPARA, NCHO are standard rules in process algebra theory; rule NSKIP states that if P_{g} evaluates to false, the execution of session $[P_{\mathrm{g}}][P_{\mathrm{rep}}] * S$ results in a NULL session; rule NRP states that if P_{g} and P_{rep} evaluate to true and the enclosed session S has an atomic action α leading to session S', then session $[P_{\mathrm{g}}][P_{\mathrm{rep}}] * S$ can evolve into $S'.\ [P_{\mathrm{rep}}] * S$ after a one-step execution of α, and, once session S' has been completed, the session $[P_{\mathrm{rep}}] * S$ evolves according to rules NRPF and NRPT.

1) Normal rules

$$\text{NSTRU: } \frac{S \equiv S' \ S \xrightarrow{p,\alpha} TT \equiv T'}{S' \xrightarrow{p,\alpha} T'}, \quad \text{NATOM: } \frac{}{\alpha \xrightarrow{\mathrm{true},\alpha} \mathrm{NULL}}, \quad \text{NSEQ1: } \frac{}{\alpha.\, S \xrightarrow{\mathrm{true},\alpha} S}, \quad \text{NSEQ2: } \frac{S \xrightarrow{p,\alpha} S'}{S.\, T \xrightarrow{p,\alpha} S'.\, T}$$

$$\text{NPARA: } \frac{S \xrightarrow{p,\alpha} S'}{S \,\|\, T \xrightarrow{p,\alpha} S' \,\|\, T}, \quad \text{NCHO: } \frac{S_1 \xrightarrow{p,\alpha} S_1'}{S_1 + S_2 \xrightarrow{p,\alpha} S_1'}, \quad \text{NRPF: } \frac{}{[p_{\mathrm{rep}}] * S \xrightarrow{\neg p_{\mathrm{rep}}, \mathrm{NULL}} \mathrm{NULL}}$$

$$\text{NRPT: } \frac{S \xrightarrow{p_{\text{resp}},\,\alpha} S'}{[p_{\text{rep}}] * S \xrightarrow{p_{\text{resp}},\,\alpha} S'.\,[p_{\text{rep}}] * S}, \quad \text{NSKIP: } \frac{}{[P_{\text{g}}][p_{\text{rep}}] * S \xrightarrow{\neg p_g,\,\text{NULL}} \text{NULL}}, \quad \text{NRP: } \frac{S \xrightarrow{p_g \wedge P_{\text{resp}},\,\alpha} S'}{[P_{\text{g}}][p_{\text{rep}}] * S \xrightarrow{p_g \wedge P_{\text{resp}},\,\alpha} S'.\,[P_{\text{rep}}] * S}$$

2) Rules for errors

$$\text{ERR-REQ: } \frac{\Gamma_{\text{Run}}, \leqslant_T \overline{\vdash} S_{\text{req}}}{S_{\text{req}} \xrightarrow[\text{true, NULL}]{} \text{Err}}, \quad \text{ERR-RESP: } \frac{\Gamma_{\text{Run}}, \leqslant_T \overline{\vdash} S_{\text{resp}}}{S_{\text{resp}} \xrightarrow[\text{true, NULL}]{} \text{Err}}, \quad \text{ERR-RR: } \frac{\Gamma_{\text{Run}}, \leqslant_T \overline{\vdash} S_{\text{req-resp}}}{S_{\text{req-resp}} \xrightarrow[\text{true, NULL}]{} \text{Err}}$$

$$\text{ERR-AS: } \frac{\Gamma_{\text{Run}}, \leqslant_T \overline{\vdash} S_{\text{assign}}}{S_{\text{assign}} \xrightarrow[\text{true , NULL}]{} \text{Err}}, \quad \text{ERR-WU: } \frac{\Gamma_{\text{Run}}, \leqslant_T \overline{\vdash} S \wedge P_g = \text{true}}{[P_{\text{g}}][P_{\text{repeat}}] * S \xrightarrow[\text{true, NULL}]{} \text{Err}}, \quad \text{ERR-SE: } \frac{S_1 \xrightarrow[\text{true, NULL}]{} \text{Err}}{S_1.\,S_2 \xrightarrow[\text{true, NULL}]{} \text{Err}}$$

$$\text{ERR-SUM: } \frac{S_1 \xrightarrow[\text{true, NULL}]{} \text{Err} \vee S_2 \xrightarrow[\text{true, NULL}]{} \text{Err}}{S_1 + S_2 \xrightarrow[\text{true, NULL}]{} \text{Err}}, \quad \text{ERR-PA: } \frac{S_1 \xrightarrow[\text{true, NULL}]{} \text{Err} \vee S_2 \xrightarrow[\text{true, NULL}]{} \text{Err}}{S_1 \,\|\, S_2 \xrightarrow[\text{true, NULL}]{} \text{Err}}$$

The rules for errors are used to capture run-time errors due to type mismatches. Rules ERR-REQ, ERR-RESP, ERR-AS, and ERR-RR state that if an atomic session is not well-typed, then run-time errors of type mismatches will be captured and the session will be led to session Err immediately; rule ERR-WU states that if an enclosed session S is not well-typed and P_{g} = true, then session $[P_{\text{g}}][P_{\text{rep}}] * S$ will be led to session Err immediately; rule ERR-SE states that if session S_1 is led to session Err due to run-time errors of type mismatches, then session $S_1.\,S_2$ will lead to session Err immediately; rules ERR-SUM and ERR-PA are similar to rule ERR-SE.

2.5 Properties of the Typed Abstract WS-CDL

The properties of the typed abstract WS-CDL are presented by the following lemmas and theorems.

Lemma 1 (*V*-weakening) If $\Gamma, \leqslant_T \vdash r[x]: \text{VT}, v, w \notin \text{Dom}(\Gamma)$, then $\Gamma, v: \text{VS}, w: C, \leqslant_T \vdash r[x]: \text{VT}$.

Proof The lemma can be proved by induction based on the depth of the derivation of $\Gamma, \leqslant_T \vdash r[x]: \text{VT}$. We consider in turn each rule as the last rule applied in the derivation:

1) If the last rule applied is TVAR, then the type assumption of $r[x]$ has been added in Γ since rule VAR has the premise $\Gamma(r[x])$ = VT. Because $v, w \notin \text{Dom}(\Gamma)$, so the premise of rule VAR still holds and we have $\Gamma, v: \text{VS}, w: C, \leqslant_T \vdash r[x]: \text{VT}$. Note that the case of 1) is a one step derivation.

2) If the derivation is an n-step derivation in which the last rule applied is TSUB, and we assume all sub derivation holds the hypothesis, so the hypothesis is true for the left premise of rule TSUB. Therefore, we have $\Gamma, v: \text{VS}, w: C, \leqslant_T \vdash r[x]: \text{VT}$.

Lemma 2 (*C*-weakening) If $\Gamma, \leqslant_T \vdash c: C_1$, and $v, w \notin \text{Dom}(\Gamma)$, then $\Gamma, v: \text{VT}, w: C_2, \leqslant_T \vdash c: C_1$.

Theorem 1 (*S*-weakening) If $\Gamma, \leqslant_T \vdash S$, and $v, w \notin \text{Dom}(\Gamma)$, then $\Gamma, v: \text{VT}, w: C, \leqslant_T \vdash S$.

Lemma 3 (*V*-narrowing) If $\Gamma,\ a: \text{VS}, \leqslant_T \vdash r[x]: \text{VT}$ and $\text{VS}' \leqslant \text{VS}$, then $\Gamma, a: \text{VS}', \leqslant_T \vdash r[x]: \text{VT}$.

Lemma 4 (*C*-narrowing) If $\Gamma, a: \text{VS}, \leqslant_T \vdash c: C$ and $\text{VS}' \leqslant \text{VS}$, then $\Gamma, a: \text{VS}', \leqslant_T \vdash c: C$.

Theorem 2 (*S*-narrowing) If $\Gamma, a: \text{VS}, \leqslant_T \vdash S$ and $\text{VS}' \leqslant \text{VS}$, then $\Gamma,\ a: \text{VS}', \leqslant_T \vdash S$.

The proof of the above lemmas and theorems is similar to the proof of lemma1, which can be accomplished by induction based on the depth of derivation of premise.

Theorem 3 (type safety) If $\Gamma, \leqslant_T \vdash S$, then for all session S' which satisfies $S \xrightarrow{P,\,\alpha} S'$, we have $\Gamma, \leqslant_T \vdash S'$.

Proof The theorem can be proved by induction based on the depth of derivation of $\Gamma, \leqslant_T \vdash S$. We consider in turn each rule as the last rule applied in the derivation:

1) If the last rule applied is TNULL, TNO, TTAU, TREQ, TRESP, TRR or TASSIGN, then S is an atomic session α. According to the rule NATOM, we have $\alpha \xrightarrow{\text{true},\,\alpha} \text{NULL}$, and $\Gamma, \leqslant_T \vdash \text{NULL}$.

2) The cases of 1) are one-step derivations. If the derivation is an n step derivation in which the last rule applied is TWORKUNIT, then the session is of the form $[P_{\text{g}}][P_{\text{rep}}] * S$ and we assume all sub derivation holds for the hypothesis. $[P_{\text{g}}][P_{\text{rep}}] * S$ has two possible derivations:

- According to rule NSKIP, we have $[P_{\text{g}}][p_{\text{rep}}] * S \xrightarrow{\neg p_g,\,\text{NULL}} \text{NULL}$, and $\Gamma, \leqslant_T \vdash \text{NULL}$.
- According to rule NRP, if $S \xrightarrow{p_g \wedge P_{\text{resp}},\,\alpha} S'$, then $[P_{\text{g}}][p_{\text{rep}}] * S \xrightarrow{p_g \wedge P_{\text{resp}},\,\alpha} S'.\,[P_{\text{rep}}] * S$. According to the induction hypothesis, S and S' are well-typed, thus we have $\Gamma, \leqslant_T \vdash S'.\,[P_{\text{rep}}] * S$.

3) Similarly, if the last rule applied is TITR, the hypothesis is also true.

4) If the derivation is an n step derivation in which the last rule applied is TSEQ, then the session is of the form $S_1.\,S_2$ and we assume all sub-derivation holds for the hypothesis. According to induction hypothesis, we have $S_1 \xrightarrow{p,\,\alpha} S_1'$ and S_1', S_2 are all well-typed. According to reduction rule NSEQ2 and typing rule TSEQ, we have $S_1.\,S_2 \xrightarrow{p,\,\alpha} S_1'.\,S_2$, and $S_1'.\,S_2$ is well typed. Similarly, if the last rule applied is TCHOICE or TPAR, the hypothesis is also true.

2.6 Operation on type assumption sets

The type assumption of a session S is denoted by Γ_S, which can be constructed in a way described by the following pseudo-code:

$$\Gamma_S = \Phi_\Gamma;\ \text{For each } r \in S[R]\ \text{For each } r[x] \in r[V]\ \ \Gamma_S = \Gamma_S, r[x]: \mathrm{VT}_x;\ \text{For each ch} \in S[C]\ \ \Gamma_S = \Gamma_S, \text{ch}: C_x$$

In the case of sequential, parallel and alternative compositions of sessions S_1, S_2, Γ_{S1} and Γ_{S2} should be merged into a larger type assumption set. The merging of Γ_{S1} and Γ_{S2}, however, is likely to induce a type conflict; i.e., variable x may have different type assumptions in Γ_{S1} and Γ_{S2}. Hence type conflicts should be eliminated when merging two type assumption sets. To express how to eliminate type conflicts, the following definitions and theorems are presented.

Definition 5 (type assumption set extension) The extension of a type assumption set Γ_S is denoted as Γ_S^{E}. Γ_S^{E} is a type assumption set satisfying:

1) $\mathrm{Dom}(\Gamma_S) \in \mathrm{Dom}(\Gamma_S^{\mathrm{E}})$;

2) $\forall x \in \mathrm{Dom}(\Gamma_S) \cap \mathrm{Dom}(\Gamma_S^{\mathrm{E}}), \leqslant_T \vdash \Gamma_S^{\mathrm{E}}(x) \leqslant \Gamma_S(x)$.

Theorem 4 (type safety holding) If $\Gamma_S, \leqslant_T \vdash S$, then $\Gamma_S^{\mathrm{E}}, \leqslant_T \vdash S$.

Proof Let $\Gamma_S^{\mathrm{E}} = \Gamma, \Delta$ and Γ, Δ satisfy

1) $\mathrm{Dom}(\Gamma) = \mathrm{Dom}(\Gamma_S)$;

2) $\forall x \in \mathrm{Dom}(\Delta), x \notin \mathrm{Dom}(\Gamma_S)$;

3) $\forall x \in \mathrm{Dom}(\Gamma),\ \Gamma(x) = \Gamma_S^{\mathrm{E}}(x)$.

Because $\Gamma_S, \leqslant_T \vdash S$ and $\forall x \in \mathrm{Dom}(\Delta), x \notin \mathrm{Dom}(\Gamma_S)$, we have $\Gamma_S, \Delta, \leqslant_T \vdash S$ by theorem 1. Moreover, for any x in $\mathrm{Dom}(\Gamma_S)$, $\Gamma_S - \{x: \Gamma_S(x)\}, x: \Gamma_S(x), \Delta, \leqslant_T \vdash S$. Considering the second condition in the definition of Γ_S^{E}, $\Gamma(x) = \Gamma_S^{\mathrm{E}}(x) \leqslant \Gamma_S(x)$; thus we have $\Gamma_S - \{x: \Gamma_S(x)\}, x: \Gamma(x), \Delta, \leqslant_T \vdash S$ according to theorem 2. After the same substitution has been made for every name in $\mathrm{Dom}(\Gamma_S)$, we have $\Gamma, \Delta, \leqslant_T \vdash S$, that is, $\Gamma_S^{\mathrm{E}}, \leqslant_T \vdash S$.

Definition 6 (compatibility of type assumption set) Two type assumption sets Γ_P, Γ_Q are compatible, denoted by $\Gamma_P \lhd \rhd \Gamma_Q$, if they satisfy either of

1) $\mathrm{Dom}(\Gamma_P) \cap \mathrm{Dom}(\Gamma_Q) = \varnothing$;

2) $\mathrm{Dom}(\Gamma_P) \cap \mathrm{Dom}(\Gamma_Q) \neq \varnothing$, and for $\forall x \in \mathrm{Dom}(\Gamma_P) \cap \mathrm{Dom}(\Gamma_Q)$, x satisfies one of ① $\leqslant_T \vdash \Gamma_P(x) \leqslant \Gamma_Q(x) \vee \leqslant_T \vdash \Gamma_Q(x) \leqslant \Gamma_P(x)$; ② $\Gamma_P(x) = R(B) \wedge \Gamma_Q(x) = W(B)$; ③ $\Gamma_Q(x) = R(B) \wedge \Gamma_P(x) = W(B)$.

Definition 7 (merging of type assumption set) If two type assumption sets Γ_P, Γ_Q satisfy $\Gamma_P \lhd \rhd \Gamma_Q$, then the merging of Γ_P, Γ_Q is denoted by $\Gamma_P \oplus \Gamma_Q$; $\Gamma_P \oplus \Gamma_Q$ is constructed in a way described by the following pseudocode:

$\Gamma_P \oplus \Gamma_Q = \Phi_\Gamma$

For each $x \notin \mathrm{Dom}(\Gamma_P) \cap \mathrm{Dom}(\Gamma_Q)$

if$(x \in \mathrm{Dom}(\Gamma_P))\ \Gamma_P \oplus \Gamma_Q = \Gamma_P \oplus \Gamma_Q, x: \Gamma_P(x)$; if $(x \in \mathrm{Dom}(\Gamma_Q))\ \Gamma_P \oplus \Gamma_Q = \Gamma_P \oplus \Gamma_Q, x: \Gamma_Q(x)$;

For each $x \in \mathrm{Dom}(\Gamma_P) \cap \mathrm{Dom}(\Gamma_Q)$

if$(\leqslant_T \vdash \Gamma_P(x) \leqslant \Gamma_Q(x))\ \Gamma_P \oplus \Gamma_Q = \Gamma_P \oplus \Gamma_Q, x: \Gamma_P(x)$; if $(\leqslant_T \vdash \Gamma_Q(x) \leqslant \Gamma_P(x))\ \Gamma_P \oplus \Gamma_Q = \Gamma_P \oplus \Gamma_Q, x: \Gamma_Q(x)$;

if $((\Gamma_P(x) = R(B) \wedge \Gamma_Q(x) = W(B)) \vee (\Gamma_Q(x) = R(B) \wedge \Gamma_P(x) = W(B)))$; $\Gamma_P \oplus \Gamma_Q = \Gamma_P \oplus \Gamma_Q, x: \mathrm{RW}(B)$

Finally, according to definition 5, $\Gamma_P \oplus \Gamma_Q$ is an extension of Γ_P, Γ_Q, and a well-typed session in Γ_P or Γ_Q is still well-typed in $\Gamma_P \oplus \Gamma_Q$.

3 Typed Rules for Mapping Choreography to Orchestration

In order to formally describe the mapping from choreography to orchestration, global interaction of web services is modeled as a session defined in typed abstract WS-CDL and local behavior of each involved role is modeled as a typed pi-calculus process[10], which is an extension of simple-typed pi-calculus proposed by Sangiorgi and Walker[11]. In addition to mapping global sessions to a local typed pi-calculus process, the type assumption set of global sessions is also needed to be mapped to the type assumption set of the local-typed pi-calculus process.

Before the mapping rules are defined, we make the following stipulations for convenience of description:

1) $|\ S\rangle_r$ denotes the local-typed pi-calculus process of role r mapped from S.

2) The mapping from proposition p to role r is denoted by p_r. For $\forall r \in p[R]$, p_r is denoted as $x_{\mathrm{pr}} = y_{\mathrm{pr}}$. Moreover, $p_r =$ true if $r \notin p[R]$.

3) The type assumption set Γ_{cho} is constructed by merging type assumption sets of all sub sessions in choreography cho. Γ_r denotes the type assumption set of the local-typed pi-calculus process of role r, $\Gamma_r = \Phi_\Gamma$ before mapping is started.

4) In order to simulate sequential execution of a pi-calculus process, we stipulate that if P is neither an input prefix nor an output prefix, then $P.Q = (\mathrm{vc})(\mathrm{vs})((P(s)\ |\ c(s).Q)$. The behavior of $P(s)$ is sending name s along internal channel c after all actions of P have been performed.

The mapping is started from top level root session and proceeds recursively according to the following rules:

1) $|\ S_{\mathrm{req}}\rangle_r = \overline{\mathrm{ch}}x, \Gamma_r = \Gamma_r \oplus \{\mathrm{ch}: \mathrm{out}(B), x: B\}$ if $(r = r_1 \wedge \Gamma_{\mathrm{cho}}, \leqslant_T \vdash r_1[x]: R(B))$
$|\ S_{\mathrm{req}}\rangle_r = \mathrm{ch}(y: B), \Gamma_r = \Gamma_r \oplus \{\mathrm{ch}: \mathrm{in}(B)\}$ if $(r = r_2 \wedge \Gamma_{\mathrm{cho}}, \leqslant_T \vdash r_2[y]: W(B))$ $|\ S_{\mathrm{req}}\rangle_r = \mathrm{skip}, \Gamma_r = \Gamma_r$ (otherwise)

2) $|S_{\text{resp}}\rangle_r = \text{ch}(x{:}B), \Gamma_r = \Gamma_r \oplus \{\text{ch}{:}\text{in}(B)\}$ if $(r = r_1 \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_1[x]{:}W(B))$

$|S_{\text{resp}}\rangle_r = \overline{\text{ch}}y, \Gamma_r = \Gamma_r \oplus \{\text{ch}{:}\text{out}(B), y{:}B\}$ if $r = r_2 \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_2[y]{:}R(B))$ $|S_{\text{req}}\rangle_r = \text{skip}, \Gamma_r = \Gamma_r$ (otherwise)

3) $|S_{\text{req-resp}}\rangle_r = \overline{\text{ch1}}x.\ \text{ch2}(s{:}B_4), \Gamma_r = \Gamma_r \oplus \{\text{ch1}{:}\text{out}(B_1), \text{ch2}{:}\text{in}(B_4), x{:}B_1\}$
if$(r = r_1 \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_1[x]{:}R(B_1) \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_1[s]{:}W(B_4))$

$|S_{\text{req-resp}}\rangle_r = \text{ch1}(y{:}B_2).\ \overline{\text{ch2}}t, \Gamma_r = \Gamma_r \oplus \{\text{ch1}{:}\text{in}(B_2), \text{ch2}{:}\text{out}(B_3), t{:}B_3\}$
if $(r = r_2 \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_2[y]{:}W(B_2) \wedge \Gamma_{\text{cho}}, \leqslant_T \vdash r_2[t]{:}R(B_3))$ $|S_{\text{req-resp}}\rangle_r = \text{skip}, \Gamma_r = \Gamma_r$ (otherwise)

4) $|S_{\text{assign}}\rangle_r = |\text{assign}(r'[x] \leftarrow r'[e])\rangle_r = \tau$ if $(r = r')$ $|S_{\text{assign}}\rangle_r = \text{skip}$ (otherwise), $\Gamma_r = \Gamma_r$

5) $|S_{\text{no}}\rangle_r = \text{skip}$ for all $r, \Gamma_r = \Gamma_r$

6) $|S_{\text{silent}}\rangle_r = |r'(\tau)\rangle_r = \tau$ if $(r = r')$ $|S_{\text{silent}}\rangle_r = \text{skip}$ (otherwise) $\Gamma_r = \Gamma_r$

7) $|S_1 \| S_2\rangle_r = |S_1\rangle_r | |S_2\rangle_r, \Gamma_r = \Gamma_r$

8) $|S_1 . S_2\rangle_r = |S_1\rangle_r . |S_2\rangle_r, \Gamma_r = \Gamma_r$

9) $|S_1 + S_2\rangle_r = |S_1\rangle_r + |S_2\rangle_r, \Gamma_r = \Gamma_r$

10) $|[P_g][P_{\text{rep}}] * S\rangle_r = [x_{\text{pgr}} = v_{\text{pgr}}](|S\rangle_r . [x_{\text{prepr}} = v_{\text{prepr}}] |[P_{\text{rep}}] * S\rangle_r)$ if $(r \in p_g[R] \wedge r \in p_{\text{rep}}[R])$

$|[P_g][P_{\text{rep}}] * S\rangle_r = [x_{\text{pgr}} = v_{\text{pgr}}](|[P_{\text{rep}}] * S\rangle_r)$ if $(r \in p_g[R] \wedge r \notin p_{\text{rep}}[R])$

$|[P_g][P_{\text{rep}}] * S\rangle_r = |S\rangle_r . [x_{\text{prepr}} = v_{\text{prepr}}] |[P_{\text{rep}}] * S\rangle_r$ if (if $r \notin p_g[R] \wedge r \in p_{\text{rep}}[R]$)

$|[P_g][P_{\text{rep}}] * S\rangle_r = |S\rangle_r . |[P_g][P_{\text{rep}}] * S\rangle_r$ (otherwise) $\Gamma_r = \Gamma_r$

11) $|[P_{\text{rep}}] * S\rangle_r = |S\rangle_r . [x_{\text{prepr}} = v_{\text{prepr}}] |[P_{\text{rep}}] * S\rangle_r$ if $(r \in p_{\text{rep}}[R])$ $|[P_{\text{rep}}] * S\rangle_r = |S\rangle_r . |[P_{\text{rep}}] * S\rangle_r$ (otherwise) $\Gamma_r = \Gamma_r$

Every rule includes two parts: 1) The mapping from a session to a typed pi-calculus process of each role; 2) The mapping from the type assumption set of global choreography to the type assumption set of the typed pi-calculus process of each role.

Because the mapping is proceeds recursively, the mapping of the type assumption set is performed only when the sessions S_{req}, S_{resp}, $S_{\text{req-resp}}$ or S_{assign} are being mapped. The type assumption set Γ_r of the pi-calculus process is constructed by $\oplus$ operator which is defined in Ref. [10].

4 Case Study

To illustrate how the operation semantics and typing rules of typed abstract WS-CDL can be used to formally reason and check the type consistency of web service choreography, let us consider the following scenario of web shopping which is depicted in Fig. 1.

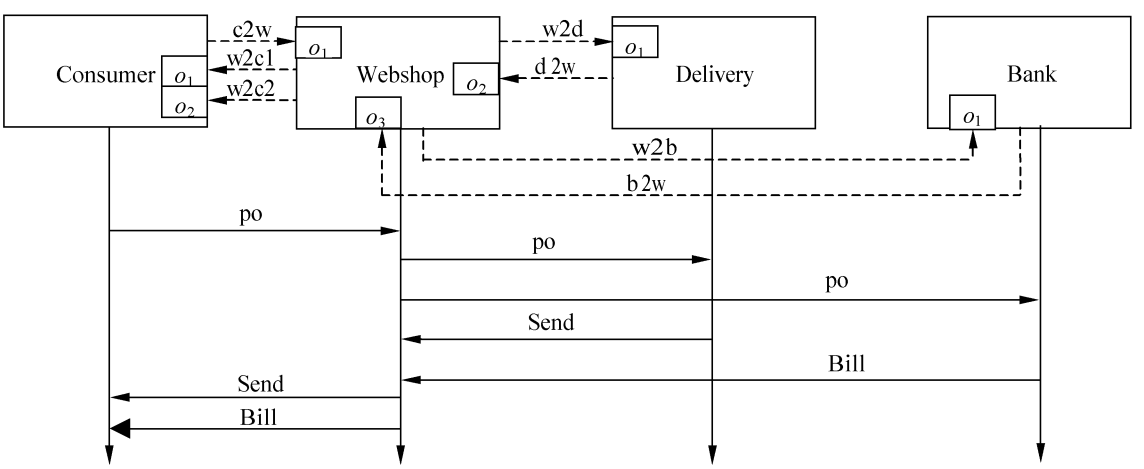

Fig. 1 Web shopping service choreography

The system is composed of four roles whose names are c, w, d and b representing the consumer, the webshop, the delivery and the bank. The consumer sends purchase order po to the webshop through channel c2w; once having received the po, the webshop initiates two parallel sessions: webshop forwards po to the delivery and bank through channels w2d and w2b; the delivery responds by returning the delivery information sent to the webshop through channel d2w while the bank responds by returning the bill to the webshop through channel b2w, and then the webshop forwards, sends and bills to the consumer through channel w2c1 and w2c2, respectively. The choreography of the four roles can be described as the following sessions:

$$\text{Order} = \text{req}(c, w, \text{c2w@}\ w.\ o_1, c[\text{po}] \rightarrow w[\text{po}])$$
$$\text{Bill} = \text{req}(w, b, \text{w2b@}\ b.\ o_1, w[\text{po}] \rightarrow b[\text{po}]).\ \text{resp}(w, b, \text{b2w@}\ w.\ o_3, w[\text{bill}] \leftarrow b[\text{bill}]).$$
$$\text{resp}(c, w, \text{w2c2@}\ c.\ o_2, c[\text{bill}] \leftarrow w[\text{bill}])$$
$$\text{Send} = \text{req}(w, d, \text{w2d@}\ d.\ o_1, w[\text{po}] \rightarrow d[\text{po}]).\ \text{resp}(w, d, \text{d2w@}\ w.\ o_2, w[\text{send}] \leftarrow d[\text{send}]).$$
$$\text{resp}(c, w, \text{w2c1@}\ c.\ o_1, c[\text{send}] \leftarrow w[\text{send}])$$

$$\text{Chor} = \text{Order}.\ (\text{Bill} \parallel \text{Send})$$

Suppose the types of data exchanged are all string and the types of channels c2w, w2c1, w2c2, w2b, b2w, w2d, d2w are $\uparrow w.\ o_1(\text{string})$, $\downarrow c.\ o_1(\text{string})$, $\downarrow c.\ o_2(\text{string})$, $\uparrow b.\ o_1(\text{string})$, $\downarrow w.\ o_3(\text{string})$, $\uparrow d.\ o_1(\text{string})$, $\downarrow w.\ o_2(\text{string})$, respectively. Thus the largest type assumption set Γ_{chor} of Chor is

$$\begin{gathered}\Gamma_{\text{chor}} = \Gamma_{\text{order}} \oplus \Gamma_{\text{bill}} \oplus \Gamma_{\text{send}} = \{\text{c2w}: \uparrow w.\ o_1(\text{string}) \\ \text{w2b}: \uparrow b.\ o_1(\text{string}),\ \text{b2w}: \downarrow w.\ o_3(\text{string}) \\ \text{w2c2}: \downarrow c.\ o_2(\text{string}),\ \text{w2d}: \uparrow d.\ o_1(\text{string}) \\ \text{d2w}: \downarrow w.\ o_2(\text{string}),\ \text{w2c1}: \downarrow c.\ o_1(\text{string}), \\ c[\text{po}]: R(\text{string}),\ w[\text{po}]: \text{RW}(\text{string}),\ b[\text{po}]: W(\text{string}) \\ w[\text{bill}]: \text{RW}(\text{string}),\ b[\text{bill}]: R(\text{string}),\ c[\text{bill}]: W(\text{string}) \\ d[\text{po}]: W(\text{string}),\ w[\text{send}]: \text{RW}(\text{string}),\ d[\text{send}]: R(\text{string}),\ c[\text{send}]: W(\text{string})\}\end{gathered}$$

Under the type assumption set Γ_{cho} and according to the typing rules defined in section 2.3, we can check the type consistency of choreography and show that Chor is a well typed session with the following inference steps:

1) The type of $w[\text{po}]$ can be inferred to have type $W(\text{string})$ under Γ_{chor} by rule TSUB since we have

$$\Gamma_{\text{chor}}, \leqslant_T \vdash w[\text{po}]: \text{RW}(\text{string}) \wedge \leqslant_T \vdash \text{RW}(\text{string}) \leqslant W(\text{string})$$

2) Session Order is well typed according to rule TREQ since we have

$$\Gamma_{\text{chor}} \vdash \text{c2w}: \uparrow w.\ o_1(\text{string}) \wedge \Gamma_{\text{chor}}, \leqslant_T \vdash c[\text{po}]: R(\text{string}) \wedge \Gamma_{\text{chor}}, \leqslant_T \vdash w[\text{po}]: W(\text{string}) \wedge \leqslant_T \vdash \text{string} \leqslant \text{string}$$

3) Similarly, the following three atomic sessions are well typed according to the rules TREQ and TRESP.

$$\begin{gathered}\text{req}(w, b, \text{w2b@ b}.\ o_1, w[\text{po}] \rightarrow b[\text{po}]) \\ \text{resp}(w, b, \text{b2w@ w}.\ o_3, w[\text{bill}] \leftarrow b[\text{bill}]) \\ \text{resp}(c, w, \text{w2c2@ c}.\ o_2, c[\text{bill}] \leftarrow w[\text{bill}])\end{gathered}$$

4) On the basis of step 3), we have $\Gamma_{\text{chor}}, \leqslant_T \vdash \text{Bill}$ by rule TSEQ.

5) Repeating inference steps 2) to 3), we have $\Gamma_{\text{chor}}, \leqslant_T \vdash \text{Send}$.

6) Finally, the choreography of the four roles is well typed by rules TPAR and TSEQ.

Moreover, we can also reason on the execution of Chor with the operational semantics of the typed abstract WS-CDL. The exhaustive inference details are not listed here anymore.

After having formally verified the execution and type consistency of Chor in the global choreography layer, we can generate the typed pi-calculus process of each involved role from Chor and verify the execution and type consistency of them in the orchestration layer. By the typed mapping rules defined in section 3, the pi-calculus processes of roles c, w, d and b (denoted as P_c, P_w, P_d and P_b respectively) are

$$\begin{gathered}P_c = \overline{\text{c2w}}\text{po}.\ (\text{w2c2}(\text{bill: string}).0 \mid \text{w2c1}(\text{send: string}).0),\ P_d = \text{w2d}(\text{po: string}).\ \overline{\text{d2w}}\text{send}.0,\ P_b = \text{w2b}(\text{po: string}).\ \overline{\text{b2w}}\text{bill}.0 \\ P_w = \text{c2w}(\text{po: string}).\ (\overline{\text{w2b}}\text{po}.\ \text{b2w}(\text{bill: string}).\ \overline{\text{w2c2}}\text{bill}.0 \mid \overline{\text{w2d}}\text{po}.\ \text{d2w}(\text{send: string}).\ \overline{\text{w2c1}}\text{send}.0)\end{gathered}$$

and the type assumption sets of P_c, P_d, P_b, P_w are

$$\begin{gathered}\Gamma_c = \{\text{c2w}: \text{out}(\text{string}),\ \text{w2c1}: \text{in}(\text{string}),\ \text{w2c2}: \text{in}(\text{string}),\ \text{po}: \text{string}\} \\ \Gamma_d = \{\text{w2d}: \text{in}(\text{string},\ \text{d2w}: \text{out}(\text{string}),\ \text{send}: \text{string}\} \\ \Gamma_b = \{\text{w2b}: \text{in}(\text{string}),\ \text{b2w}: \text{out}(\text{string}),\ \text{bill}: \text{string}\} \\ \Gamma_w = \{\text{c2w}: \text{in}(\text{string}),\ \text{w2b}: \text{out}(\text{string}),\ \text{b2w}: \text{in}(\text{string}),\ \text{send}: \text{string},\ \text{bill}: \text{string},\ \text{w2c1}: \text{out}(\text{string}), \\ \text{w2c2}: \text{out}(\text{string}).\ \text{w2d}: \text{out}(\text{string}),\ \text{d2w}: \text{in}(\text{string}),\ \text{po}: \text{string}\}\end{gathered}$$

According to the definition of $\oplus$ operator on the type assumption set and the typing rules for the typed pi-calculus that we have proposed in Ref. [10], we can infer that P_c, P_d, P_b, P_w are all well typed under type assumption set $\Gamma_{\max}$ ($\Gamma_{\max} = \Gamma_c \oplus \Gamma_d \oplus \Gamma_b \oplus \Gamma_w$). The local behavior of P_c, P_d, P_b, P_w can be verified by MWB, a tool for manipulating and analyzing mobile concurrent systems described in the pi-calculus[12]. Using the deadlocks and step commands in MWB, we determine that the parallel composite process $P_c \mid P_d \mid P_b \mid P_w$ has no deadlocks and can successfully terminate.

Finally, let us consider the cases of capturing run-time errors due to type mismatches. Suppose that the type of po sent from the consumer to the webshop is int, and then by rule ERR-REQ and ERR-SE, we have

$$\dfrac{\dfrac{\dfrac{\Gamma_{\text{chor}}, \leqslant_T \vdash \text{c2w}: \uparrow w.\ o_1(\text{string}) \wedge \Gamma_{\text{chor}}, \leqslant_T \vdash c[\text{po}]: R(\text{int}) \wedge \Gamma_{\text{chor}}, \leqslant_T \vdash w[\text{po}]: w(\text{string}) \wedge \leqslant_T \vdash \text{int} \overline{\leqslant} \text{string}}{\Gamma_{\text{chor}}, \leqslant_T \overline{\vdash}\ \text{req}(c, w, \text{c2w@ } w.\ o_1, c[\text{po}] \rightarrow w[\text{po}])}}{\text{req}(c, w, \text{c2w@ } w.\ o_1, c[\text{po}] \rightarrow w[\text{po}]) \xrightarrow{\text{NULL}} \text{Err}}}{\text{Chor} \xrightarrow{\text{NULL}} \text{Err}}$$

that is, the run-time errors due to type mismatches can be captured by the rules defined in section 2.4.

5 Conclusion

In this paper we propose a typed formal model for WS-CDL specification. Within the model, the operation semantics can help to reason the execution of web service choreography and capture run-time errors due to type mismatches while the typing rules can check the type consistency. The properties of our model are given and proved. The typed rules for mapping choreography to orchestration are defined so that web services composition can be verified in choreography and orchestration levels, respectively.

References

[1] Bucchiarone A, Gnesi S. A survey on services composition languages and models [C]//*Proc of the International Workshop on Web Services Modeling and Testing*. Palermo, Italy, 2006: 51 - 63.

[2] Kavantzas N, Burdett D, Ritzinger G, et al. Web service choreography description language version 1.0 [EB/OL]. (2005-11-09) [2008-04-01]. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

[3] Salaun G, Bordeaux L, Schaerf M. Describing and reasoning on web services using process algebra [C]//*Proc of the 2nd IEEE International Conference on Web Services*. Washington, DC: IEEE Computer Society Press, 2004: 43 - 50.

[4] Brogi A, Canal C, Pimentel E, et al. Formalizing web service choreographies [J]. *Electronic Notes in Theoretical Computer Science*, 2004, **105**: 73 - 94.

[5] Busi N, Gorrieri R, Guidi C, et al. Towards a formal framework for choreography [C]//*Proc of the 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises.* Washington, DC: IEEE Computer Society Press, 2005: 107 - 112.

[6] Yeung W L, Wang J, Dong W. Verifying choreographic descriptions of web services based on CSP [C]//*Proc of the IEEE Services Computing Workshops(SCW'06)*. Washington, DC: IEEE Computer Society Press, 2006: 97 - 104.

[7] Zhao Xiangpeng, Yang Hongli, Qiu Zongyan. Towards the formal model and verification of web service choreography description language [C]//*Proc of the 3rd International Workshop on Web Service and Formal Methods*. Springer-Verlag, 2006: 273 - 287.

[8] Gay S, Hole M. Subtyping for session types in the pi calculus [J]. *Acta Informatica*, 2005, **42**(2): 192 - 225.

[9] Pahl C. A pi-calculus based framework for the composition and replacement of components [C]//*Proc of Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA'2001)—Workshop on Specification and Verification of Component-Based Systems*. New York: ACM Press, 2001: 97 - 107.

[10] Gu Xiwu, Lu Zhengding. A typed formal model for web services composition [J]. *Computer Science*, 2008, **35**(1): 128 - 134.

[11] Sangiorgi D, Walker D. *The pi-calculus: a theory of mobile process* [M]. Cambridge: Cambridge University Press, 2001: 231 - 308.

[12] Victor B, Moller F. The mobility workbench—a tool for the pi-calculus [C]//*Proc of the 6th International Conference on Computer Aided Verification*. Springer-Verlag, 1994: 428 - 440.

Web 服务组合规范 WS-CDL 的类型化形式化模型

辜希武　李瑞轩　卢正鼎

(华中科技大学计算机科学与技术学院,武汉 430074)

摘要:为了形式化地推理和验证 web 服务编排规范 WS-CDL 所描述的 web 服务组合,提出了一个 WS-CDL 规范的类型化形式化模型——typed abstract WS-CDL. 在 typed abstract WS-CDL 中,定义了类型和会话的语法、类型判定规则和操作语义;web 服务间的协作由会话来描述;利用会话的操作语义能对 web 服务编排的执行进行推理;利用类型判定规则能检查 web 服务间交换信息类型一致性并捕获由于类型不一致导致的运行时错误. 特别地提出了类型假设集的外延和类型假设集相容性的概念,并定义了类型假设集的合并算法以消除类型假设冲突. 在该模型基础上,还定义了从 choreography 到 orchestration 的类型化映射规则,通过这组规则,可以从一个给定的 web 服务 choreography 得到 orchestration 桩代码及其类型假设集,因而 web 服务组合能在 choreography 和 orchestration 层被验证. 提出的模型被证明具有类型安全性,并且通过一个案例分析说明了所提出的模型是有助于对 web 服务组合进行推理和验证的.

关键词:类型化模型;web 服务组合;web 服务编排描述语言

中图分类号:TP311