Research on Component Composition and Replacement with Formal Semantics

Ruzhi Xu, Quansheng Wu, Peiguang Lin+
School of Information Engineering, Shandong University of Finance, Jinan, China, 250014
Email: llpwgh@sdfi.edu.cn

***Abstract*—Based on the notion of 'design by contract', components interaction patterns and process patterns of component composition, formal semantics of components are proposed. With this basis and inspired by typing system and process construction methods in π -calculus, in this paper, a formal typing framework for the composition and replacement of components are proposed. Additionally, rules about component composition and replacement are introduced based on component operation semantics and π -calculus typing rules, which establish a foundation for rigorously analyzing and reasoning the composed system.**

***Index Terms*: Component-based Software Engineering (CBSE); component composition; component replacement; design by contract; π -calculus**

I. INTRODUCTION

Component composition & replacement are considered as key technology and research focus in CBSE[1]. Construction and evolution of target system can be achieved through component composition & replacement.

At first, formal semantics of composition is proposed in this paper based on design by contract [2], component interaction [3] and process of component composition [4]. Contract semantics composed of signature, pre-condition and post condition is used to describe computation characteristics of component and is also key element of component match and correctness validation; Behavior semantics expressed by component interaction model is the foundation of analysis, reasoning and validation towards component composition and replacement ; Operational semantics expressed by process model is the foundation of formal modeling towards component composition and replacement.

Formal type framework of component composition and replacement is proposed based on formal semantics by referencing type system and process construction in π process calculus [5]. This type framework not only provides deep analysis to port which is an important entity of component composition, but also differentiates port type and channel type of each type. Therefore different roles played by different ports in the process of component composition can be distinguished.

Rules for component composition and replacement are provided based on operational semantics and type rules of π calculus. Rigorous analysis and reasoning are done towards composition correctness and influence range of replacement. Run-time errors can be avoided to enhance robustness and maintainability of target system.

In this paper, formal semantics is introduced firstly; secondly typing system of replacement is modeled combining with π calculus and described in detail; thirdly rules for composition and replacement are introduced; finally introduction and summary for corresponding research are presented.

II. FORMAL SEMANTICS OF COMPONENT

Traditional component service description mainly includes syntax information such as function name and parameter list. Formal semantics [6] must be included in component description in order to rigorously analyze and reason component composition and replacement. There are contract, behavior and operational semantics in component semantics

A. Component Contract Semantics

Calculation characteristics are described by component contract, which is denoted as TCRT (TSIG, PRE, POST) and is composed of component service's framework, pre-condition and post-condition. Framework mainly described syntax information such as function name and parameter lists; Pre-condition is the condition which must be satisfied in order to implement component service, otherwise results of service implementation will be uncertain.; Post-condition is the condition which will be established after implementing component service.

Definition 1 (Contract) let A be some kind of operation (an instruction or a function), when A is arbitrarily implemented from the state in which M is established, A will terminate to the state in which N is also established, {M} A {N} is invoked as a contract in which M is invoking pre-condition and N is invoking post-condition.

For example , contract semantics for service put() of component Stack is described as follows, in which pre and post represent pre-condition and post-condition of service respectively:

```
Component STACK [G]
   put (x: G) is        -- Add x on top
     pre   not full
     post not empty & item = x & count = old count + 1
   end
end
```


doi:10.4304/jsw.6.9.1640-1648

B. Component Behavior Semantics

Component interacts with other components by the means of providing and requesting services. If service provided by one component satisfies service requested by another component, we can compose these two components. Based on different models of component interaction, different behavior semantics will be exhibited by composed component. Let *R* and *S* be two components:

Definition 2 (Serial Interaction) Serial interactions of components *R* and *S* is denoted as *R*;*S* and its behavior semantics is that operation of *S* followed operation of *R*.

Definition 3 (Uncertainty Selective Interaction) Uncertainty selective interaction of components *R* and *S* is denoted as *R*+*S* and its behavior semantics is that only one operation, either *R* or *S*, can be executed based on a specific context state.

Definition 4(Repeated Interaction) Repeated interaction of component *R* is denoted as !*R* and its behavior semantics is that *R* will be executed repeatedly.

Definition 5 (Parallel Interaction) Parallel interactions of components *R* and *S* is denoted as *R* |*S* and its behavior semantics is that *R* and *S* will be executed in parallel.

The following expression of component interaction process can be obtained by combining process construction methods of π -calculus:

$$R ::= R_1;R_2 \mid R_1 + R_2 \mid !R \mid R_1 | R_2 \mid (v\,a)R \mid 0$$

$(v\,a)R$ represents that variable *a* can be only visible in *R*, i.e. *a* is private variable of process *R*. 0 represents inaction, i.e. process without executing any action. While $R\{b/a\}$ represents that variable *a* in *R* is substituted by variable *b*.

C. Component Operational Semantics

Component composition includes multiple stages such as service matching and establishing of interaction channel, invoking and execution of service, as well as service replying and result requesting. Let *R* (Requestor) and *S* (Server) represent two components of requesting and providing service, so there will be the following process model in component composition:

(1) Service match: detect if service *s* provided by *S* can satisfy service *r* requested by *R*; (2)Establishment of interaction channel: When *R* is invoking the service of *S*, a interaction channel *c* must be established between port *r* and *s*;(3)Service invoking: *R* sends request for invoking service of *S* through interaction channel and agrees to establish reply channel simultaneously;(4)Service execution: Corresponding service will be executed after *S* receives invoking request from *R* through interaction channel ;(5)Service reply: *S* sends corresponding reply information through reply channel after executing requested service;(6)Service result: *R* receives execution results of service from reply channel.

In π -calculus , basic element for describing behavior is action $\alpha ::= \mathrm{T}_{PORT}\,\overline{x}\langle y\rangle \mid \mathrm{T}_{PORT}\,x(y) \mid \tau$ .Where T_{PORT} $\overline{x}\langle y\rangle$ represents type of active port, output action represents sending name y through channel (port) x, input action $x(y)$ represents receiving name y from x, unobservable action τ is inner action in process and is not visible outside process.

Combining with the process pattern of component composition, behavior semantics displayed in the process of component composition by various ports is shown as following prefix:

$$\pi ::= \mathrm{T}_{REQ}\,\overline{r_C}\langle r_I\rangle \mid \mathrm{T}_{SER}\,s_C(s_I) \mid \mathrm{T}_{INV}\,\overline{r_I}\langle a_1,...,a_s,r_R\rangle \mid \mathrm{T}_{EXE}\,s_I(x_1,...,x_t,s_R) \mid \mathrm{T}_{REP}\,\overline{s_R}\langle b\rangle \mid \mathrm{T}_{RES}\,r_R(y)$$

Where T_{REQ} represents service requested, T_{SER} represents service provided, T_{INV} represents service invoked, T_{EXE} represents service executed, T_{REP} represents service replied, and T_{RES} represent result of service.

III. Typing Framework of Component Composition and Replacement

The major use of typing system is to prevent errors during operation [5, 7, 8, 9], and it is key factor to analyze and reason the correctness of component composition. Combining with the typing system of π -calculus, the following type judgments are introduced in this paper. Where $\Gamma = \{v_1 : \mathrm{T}_1,...,v_n : \mathrm{T}_n\}$ called type environment and it is gives type T_i to name, while each v_i appears in Γ only once:

$\Gamma \vdash \mathrm{P}$ Expression P is well-typedness, i.e. every variable in P is defined in Γ

$\Gamma \vdash x : \mathrm{T}$ Name x belongs to type T

$\Gamma \vdash \mathrm{T} \le \mathrm{S}$ Type T is subtype of type S

Based on characteristics of component composition and formal semantics of component, the following typing framework is introduced in this paper:

$\mathrm{T} ::= \mathrm{T}_{BSC}$ Basic type

$\mid \mathrm{T}_{SIG}(\mathrm{T}\times...\times\mathrm{T}\times\mathrm{T}_{LNK})$ Signature

$\mid \mathrm{T}_{PRD}(\mathrm{T})$ Predicate

$\mid \mathrm{T}_{LNK}$ Link type

$\mathrm{T}_{LNK} ::= \mathrm{T}_C \;\; \mathrm{T}_P$ Channel and Port type

$\mathrm{T}_C ::= \mathrm{T}_{CTR}(\mathrm{T}\times\mathrm{T}\times\mathrm{T})$ Contract

$\mid \mathrm{T}_{INT}(\mathrm{T}\times...\times\mathrm{T}\times\mathrm{T}_{LNK})$ Interaction

$\mid \mathrm{T}_{RLY}(\mathrm{T})$ Reply

$\mathrm{T}_P ::= \pm(\mathrm{T}_{REQ} \mid \mathrm{T}_{SER} \mid \mathrm{T}_{INV} \mid \mathrm{T}_{EXE} \mid \mathrm{T}_{REP} \mid \mathrm{T}_{RES})$ Port type

Where a group T_{BSC} representing basic types such as Integer and String is supposed. T_{SIG} and T_{PRD} belong to

basic type representing signature and predicate of service. T_{CTR},T_{INT} and T_{RLY} all belong to link type and they are classified by data type transferred through the channel.

Component composition usually involves in three entities including data, port and component and port representing component service is the most important one. p=(p_{CTR}, p_{INT}, p_{REP}) is used to represent component port, where p_{CTR} is the contract port defined by type structure, pre- and post-condition, p_{INT} is interactive port to implement service invoking and execution, p_{REP} is reply port to transfer replied information of service.

To differentiate different roles played by various ports during the process of component interaction, port type & channel type of each port will be classified in this paper.

$T_p(p)$ or $p :_p \mathrm{t}$ is used to represent the port type of port p. It describes the functional and directional characteristics of component port. Functional characteristics is described by contract port and interactive port, while directional characteristics refers to input or output port.

Action prefix describing behavior semantics of component in the previous section represents port type. For example, $\overline{r_C}$ is service request port, while s_C is its paired port for service providing, i.e. $T_{REQ} = T_p(\overline{r_C})$, $T_{SER} = T_p(s_C)$. They can also be represented as $\overline{r_C} :_p T_{REQ}$ and $s_C :_p T_{SER}$.

Each port has one direction and it is called polar. "+" represents output, i.e. this port can only be used to send information; "−" represents input, i.e. this port can only be used to receive information.

Each group of ports(p_{CTR}, p_{INT}, p_{REP}) usually follows certain direction model. For example, $(+ + -)$ represents request port, i.e. send contract and interaction messages, and receive reply message via interactive channel; $(- - +)$ represents server port, i.e. receive contract and interaction messages, and send reply message via interactive channel.

$T_c(p)$ or $p :_c \mathrm{t}$ is used to represent channel type for port p and it describes capability of transferring entity through port.

Channel type can restrain interaction and composition between components, while $p_{\mathrm{CTR}} :_c T_{CTR}(T_{SIG}(T_1,\ldots,T_n, T_{RLY}(T)), T_{PRD}(\mathrm{PRE}), T_{PRD}(\mathrm{POST}))$ represent contract types that must be satisfied by interaction channel when it is transferred through contract type. Where $p_{\mathrm{INT}} :_c T_{INT}(T_1,\ldots,T_n, T_{RLY}(T))$ means that data and reply information can be delivered by interaction port, $p_{\mathrm{REP}} :_c T_{RLY}(T)$ means that only data can be carried by reply port. Framework of contract port $T_{SIG}(T_1,\ldots,T_n, T_{RLY}(T))$ represents reply channel $T_{RLY}(T)$ is built at the same time when delivering parameters $T_1,\ldots,T_n$ by invoking component service. Pre- and post-conditions are represented by predicate type T_{PRD}. Channel types of each port during component composition process are shown as follows:

$$r_C \ : \ T_{CTR}\left(T_{SIG}\left(T_1,\ldots,T_n,T_{RLY}(T)\right),T_{PRD}(\mathrm{PRE}),T_{PRD}(\mathrm{POST})\right)$$

$$s_C \ : \ T_{CTR}\left(T_{SIG}\left(T_1',\ldots,T_n',T_{RLY}(T')\right),T_{PRD}(\mathrm{PRE}'),T_{PRD}(\mathrm{POST}')\right)$$

$$r_I \ : \ T_{INT}\left(T_1,\ldots,T_n,T_{RLY}(T)\right)$$

$$s_I \ : \ T_{INT}\left(T_1',\ldots,T_n',T_{RLY}(T')\right)$$

$$r_R \ : \ T_{RLY}(T) \qquad\qquad s_R \ : \ T_{RLY}(T')$$

Where interaction and reply ports can be only visible by two components involved in interaction, i.e. the interaction channel built during component composition is private channel between components.

IV. Rules for Component Composition

Typing rules are used to define semantics of typing system in π -calculus [5]. Based on typing rules, effectiveness of one specific judgment can be judged on the basis of known effected judgments. The format for typing rule is as follows:

$$[\text{Rule Name}] \quad \frac{\Gamma_1 \vdash \mathbb{S}_1 \quad \ldots \quad \Gamma_n \vdash \mathbb{S}_n}{\Gamma \vdash \mathbb{S}} \langle \ \text{side-condition}$$

Here, each rule has a name determined by its conclusion. The part above horizontal line is multiple hypotheses of rule $\Gamma_i \vdash \mathbb{S}_i$, while the part below it is rule's only conclusion $\Gamma \vdash \mathbb{S}$ .The part after $\langle$ is the optional side-condition. The meaning of rule is that conclusion is valid when all hypotheses are satisfied.

A. Basis Typing Rules

Following are some basic typing rules combined with typing framework of component composition:

Rule 1 (Sub-typing Relation Rule) Sub-typing relation $\leq$ is a preorder relation, i.e. it has reflexivity and transitivity:

$$[\mathrm{SUB_{REFL}}] \quad \frac{T = S}{\Gamma \vdash T \leq S}$$

$$[\mathrm{SUB_{TRANS}}] \quad \frac{\Gamma \vdash U \leq T \quad \Gamma \vdash T \leq S}{\Gamma \vdash U \leq S}$$

Rule 2 (Reply Channel Sub-typing Rule) If data type T' delivered by reply channel R' is the subtype of T delivered by reply channel R, then type $T_{RLY}(T')$ for reply channel R' is the subtype of $T_{RLY}(T)$ for reply channel R:

$$[\mathrm{SUB_{RLY}}] \quad \frac{\Gamma \vdash T' \leq T}{\Gamma \vdash T_{RLY}(T') \leq T_{RLY}(T)}$$

Rule 3(Contract Typing Rule) If types of s, p_1, p_2 are framework type, pre-condition type and post-condition type, then the contract type of contract T_{CTR} (s, p_1, p_2) is $T_{CTR}\left(T_{SIG}\left(T_1,\ldots T_n,T_{RLY}(T)\right),T_{PRD}(T),T_{PRD}(T)\right)$:

$$[\mathrm{IS_{CTR}}] \quad \frac{\Gamma \vdash s :_c T_{SIG}(T_1,\ldots T_n,T_{RLY}(T)) \quad \Gamma \vdash p_1 :_c T_{PRD}(T) \quad \Gamma \vdash p_2 :_c T_{PRD}(T)}{\Gamma \vdash T_{CTR}(s,p_1,p_2) :_c T_{CTR}(T_{SIG}(T_1,\ldots T_n,T_{RLY}(T)),T_{PRD}(T),T_{PRD}(T))}$$

Rule 4(Framework Sub-typing Rule) If all data types in framework S' are subtypes of corresponding data types in framework S, and type for reply channel in S is the

subtype of that for reply channel in *S'*, then framework type for *S'* is the subtype of that for *S*:

$$[\mathrm{SUB_{SIG}}]\quad \frac{\Gamma \vdash \mathrm{T_1}' \le \mathrm{T_1} \quad ... \quad \Gamma \vdash \mathrm{T_n}' \le \mathrm{T_n} \quad \Gamma \vdash \mathrm{T_{RLY}(T)} \le \mathrm{T_{RLY}(T')}}{\Gamma \vdash \mathrm{T_{SIG}(T_1',...,T_n',T_{RLY}(T')) \le T_{SIG}(T_1,...,T_n,T_{RLY}(T))}}$$

Rule 5(Predicate Sub-typing Rule) If condition COND' implicates condition COND, then predicate type for COND' is the subtype of predicate type COND:

$$[\mathrm{SUB_{PRD}}]\quad \frac{\mathrm{COND}' \to \mathrm{COND}}{\Gamma \vdash \mathrm{T_{PRD}(COND') \le T_{PRD}(COND)}}$$

Rule 6 (Contract Sub-typing Rule) If contract *C'* is the contract obtained after weakening pre-condition and strengthening post-condition of contract *C*, and the framework type for *C'* is the subtype of that for *C*, then the contract type for *C'* is the subtype of that for *C*:

$$[\mathrm{SUB_{CTR}}]\quad \frac{\Gamma \vdash \mathrm{PRE} \le \mathrm{PRE}' \quad \Gamma \vdash \mathrm{POST}' \le \mathrm{POST} \quad \Gamma \vdash \mathrm{T_{SIG}}' \le \mathrm{T_{SIG}}}{\Gamma \vdash \mathrm{T_{CTR}(T_{SIG}',PRE',POST') \le T_{CTR}(T_{SIG},PRE,POST)}}$$

Rule 7 (Sub-typing Rule for Interaction Channel) If all data types in interaction channel *I'* are subtypes of corresponding data types in interaction channel *I*, and type for reply channel in *I* is the subtype of that for reply channel in *I'*, then interaction channel type for *I'* is the subtype of that for *I*:

$$[\mathrm{SUB_{INT}}]\quad \frac{\Gamma \vdash \mathrm{T_1}' \le \mathrm{T_1} \quad ... \quad \Gamma \vdash \mathrm{T_n}' \le \mathrm{T_n} \quad \Gamma \vdash \mathrm{T_{RLY}(T)} \le \mathrm{T_{RLY}(T')}}{\Gamma \vdash \mathrm{T_{INT}(T_1',...,T_n',T_{RLY}(T')) \le T_{INT}(T_1,...,T_n,T_{RLY}(T))}}$$

B. Service Matching and Well-typedness

Above basic typing rules can be used to represent sub-typing relation among all entities (data, port and component) involved during component composition. Sub-typing relation can be used to judge not only whether two service ports match with each other or not, but also if they can be composed together.

During component composition, match, i.e. service provided can satisfy requested service, need to be judged firstly. Type for contract channel composes of framework, pre- and post-condition. Based on traditional refinement relation(weakening pre-condition and strengthening post-condition), match is defined as follows:

Definition 6 (Match) For server $r_C :_c \mathrm{T_{CRT}(T_{SIG}, PRE, POST)}$ and request $s_C :_c \mathrm{T_{CRT}(T_{SIG}', PRE', POST')}$, if $\mathrm{T_{SIG} = T_{SIG}}' \wedge \mathrm{PRE} \to \mathrm{PRE}' \wedge \mathrm{POST}' \to \mathrm{POST}$, then server s_C matches r_C.

The implied condition for this definition is: $\mathrm{T_{REQ}}$ and $\mathrm{T_{SER}}$ are required to be complementary and have opposite port direction when they are matched. It is denoted as $\mathrm{T}(r_C) \Leftrightarrow \mathrm{T}(s_C)$.

Interaction channel between components will be established after matching and the type of delivered entity must be type allowed by the channel while delivering information along this channel, i.e. type satisfaction.

Definition 7 (Interaction Typing Satisfies Contract Typing) For interaction typing $\mathrm{T}_I = \mathrm{T}_{INT}(\mathrm{T}_1,...,\mathrm{T}_n,\mathrm{T}_{RLY}(\mathrm{T}))$ and contract typing $\mathrm{T}_C = \mathrm{T}_{CTR}(\mathrm{T}_{SIG},\mathrm{PRE},\mathrm{POST})$, if interaction port s_I of service s satisfies restraint $\mathrm{T_{SIG}} = \mathrm{T_{SIG}}(\mathrm{T}_1,...,\mathrm{T}_n,\mathrm{T}_{RLY}(\mathrm{T}))$, and if the pre-condition PRE is valid, the post-condition POST will be valid after the execution of s_I, we call interaction typing T_I satisfies contract typing T_C, and denote it as $\mathrm{T}_I \models \mathrm{T}_C$.

In the meantime, based on description of port channel typing, successful interaction between components also requires that reply typing T_R must satisfy interaction typing T_I and data typing T must satisfy reply typing T_R, i.e. $\mathrm{T}_R \models \mathrm{T}_I$ and $\mathrm{T} \models \mathrm{T}_R$.

Based on typing framework of component composition, typing judgment $\Gamma \vdash \mathrm{E}$ represents that all variables in expression E have definitions under the typing environment Γ, i.e. E has well-typedness. The fundamental effect of well-typedness lies in : Once composition service contract is established on the basis of typing framework for component composition, it can guarantee correct behavior of composition and interaction. Well-typedness has the following attributes (No concrete proof will be given since they are relatively simple):

Lemma 1(Judgment in program with well-typedness will not fail) If $\Gamma \vdash \mathrm{R}$, then execution of R will not fail.

Lemma 2 (Variable substitution can keep well-typedness) If $\Gamma \vdash \mathrm{R}$, and $\Gamma \vdash x:\mathrm{T}, v:\mathrm{T}$, then $\Gamma \vdash \mathrm{R}\{v/x\}$.

Lemma 3(Transition can keep well-typedness) If $\Gamma \vdash \mathrm{R}_1$ and $\mathrm{R}_1 \to \mathrm{R}_2$, then $\Gamma \vdash \mathrm{R}_2$.

Combined with typing satisfaction, well-typedness for component composition is defined as follows:

Definition 8 (Well-typedness for Component Composition)

(1) If $\mathrm{T}_c(r_I) \models \mathrm{T}_c(r_C)$, then $\Gamma \vdash \mathrm{T_{REQ}}\ \overline{r_C}\langle r_I \rangle$, otherwise $\mathrm{T_{REQ}}\ \overline{r_C}\langle r_I \rangle$ fails.

(2) If $\mathrm{T}_c(s_I) \models \mathrm{T}_c(s_C)$, then $\Gamma \vdash \mathrm{T_{SER}}\ s_C(s_I)$, otherwise $\mathrm{T_{SER}}\ s_C(s_I)$ fails.

(3) If a is data type, and $\mathrm{T}_c(r_R) \models \mathrm{T}_c(r_I)$, then $\Gamma \vdash \mathrm{T_{INV}}\ \overline{r_I}\langle a, r_R \rangle$, otherwise $\mathrm{T_{INV}}\ \overline{r_I}\langle a, r_R \rangle$ fails.

(4) If y is data type, and $\mathrm{T}_c(s_R) \models \mathrm{T}_c(s_I)$, then $\Gamma \vdash \mathrm{T_{EXE}}\ s_I(y, s_R)$, otherwise $\mathrm{T_{EXE}}\ s_I(y, s_R)$ fails.

Here, execution of action fails means that data delivered through channel violates typing constraint of channel, interaction fails means that two action involved in interaction are well-typedness but they violates typing constraint of channel.

Successful parallel composition must satisfy the following well-typedness rule based on interaction semantics of component and the concept of well-typedness

Rule 8 (Well-typedness Rule for Parallel Composition)

$$[\mathrm{WELL_{PAR\text{-}COMP}}]\quad \frac{\Gamma \vdash \mathrm{T_{REQ}}\ \overline{r_C}\langle r_I \rangle \quad \Gamma \vdash \mathrm{T_{PRO}}\ p_C(p_I) \quad \Gamma \vdash \mathrm{T}_c(p_C) \le \mathrm{T}_c(r_C)}{\Gamma \vdash \mathrm{T_{REQ}}\ \overline{r_C}\langle r_I \rangle \mid \mathrm{T_{PRO}}\ p_C(p_I)}$$

If $\Gamma \vdash \mathrm{T_{REQ}}\ \overline{r_C}\langle r_I \rangle$ and $\Gamma \vdash \mathrm{T_{SER}}\ s_C(s_I)$ all are valid, but $\Gamma \vdash \mathrm{T}_c(s_C) \le \mathrm{T}_c(r_C)$ is not valid, then parallel composition fails.

C. Transition Rule for Component Composition

In order to understand dynamic activity of system, action is used to mark transition relation between processes in π calculus[12]. Transition relation marked by action α is denoted as $\xrightarrow{\alpha}$. Transition $P \xrightarrow{\overline{x}\langle y\rangle} Q$ means that process *P* evolves into process *Q* after sending name *y* through channel (port) *x*; Transition $P \xrightarrow{x(y)} Q$ means that process *P* evolves into process *Q* after receiving *y* from *x*; Transition $P \xrightarrow{\tau} Q$ means process *P* can evolve into process *Q* without being visible from outside.

Combined with process pattern for component composition, we built modeling on dynamic activity using transition relation as follows.

From the definition of match, a sub-typing relation is formed when service *s* served by component *S* matches service *r* requested by component *R*. At this moment, an interaction channel will be established between these two components and service interaction occurs through this channel. Match transition rule is used to represent as:

Rule 9 (Match Transition Rule)

$$[\mathrm{TRANS_{MAT}}]\quad \frac{\mathrm{T}_{REQ}\ \overline{r_C}\langle r_I\rangle.R \xrightarrow{\overline{r_C}\langle r_I\rangle} R \quad \mathrm{T}_{PRO}\ p_C(p_I).P \xrightarrow{p_C(p_I)} P}{\mathrm{T}_{REQ}\ \overline{r_C}\langle r_I\rangle.R \mid \mathrm{T}_{PRO}\ p_C(p_I).P \xrightarrow{\tau} R*P}\langle\Theta$$

Here, additional condition is $\Theta \equiv \mathrm{T}_{s_C} \le \mathrm{T}_{r_C}$ and its connection $R*S \stackrel{\mathrm{def}}{=} v\,c\left(R\{c/r_I\} \mid S\{c/s_I\}\right)$ introduces a new variable *c*, which represents the private interaction channel built between *R* and *S* while matching.

After private interaction channel *c* is built between *S* and *R*, component *R* sends parameter *a: t_a* (satisfy $\mathrm{T}_a \le \mathrm{T}_x$) to this channel and reply channel $\mathrm{r}_R : \mathrm{T}_{r_R}$ invoke service s_I from *S*. It can be represented using Interaction Transition Rule as follows:

Rule 10(Interaction Transition Rule)

$[\mathrm{TRANS_{INT}}]$

$$\frac{\mathrm{T}_{INV}\ \overline{r_I}\langle a, r_R\rangle.R \xrightarrow{\overline{r_I}\langle a, r_R\rangle} R \quad \mathrm{T}_{EXE}\ p_I(x, p_R).P \xrightarrow{p_I(x, p_R)} P}{\mathrm{T}_{INV}\ \overline{r_I}\langle a, r_R\rangle.R \mid \mathrm{T}_{EXE}\ p_I(x, p_R).P \xrightarrow{\tau} R*P\{a/x\}}\langle\Theta$$

Here, additional condition is $\Theta \equiv \mathrm{T}_{s_I} \le \mathrm{T}_{r_I}$, parameter *a* : PRE(*a* must satisfy pre-condition). T_{r_I} and T_{s_I} are interaction type $\mathrm{T}_{INT}\left(t_1,...,t_r,\mathrm{T}_{RLY}(t)\right)$ and $\mathrm{T}_{INT}\left(t_1{}',...,t_p{}',\mathrm{T}_{RLY}(t')\right)$ respectively and their reply channel (type T_{RLY}) is a private channel between *R* and *S*.

After completing service interaction, component *S* will send back reply information *b : t_b*(satisfy $\mathrm{T}_b \le \mathrm{T}_y$) to component *R* through reply channel. It can be represented using Reply Transition Rule as follows:

Rule 11(Reply Transition Rule)

$[\mathrm{TRANS_{RLY}}]$

$$\frac{\mathrm{T}_{RES}\ r_R(y).R \xrightarrow{r_R(y)} R \quad \mathrm{T}_{REP}\ \overline{p_R}\langle b\rangle.P \xrightarrow{\overline{p_R}\langle b\rangle} P}{\mathrm{T}_{RES}\ r_R(y).R \mid \mathrm{T}_{REP}\ \overline{p_R}\langle b\rangle.P \xrightarrow{\tau} R\{b/y\}*P}\langle\Theta$$

Here, additional condition is $\Theta \equiv \mathrm{T}_{s_R} \le \mathrm{T}_{r_R}$, return value *b* : POST(i.e. *b* must satisfy post-condition) is the inner calculation result of service *s*.

D. Formal Model of Component

Action prefix[5] $\pi.P$ in π-calculus means that process *P* can be executed only after executing action π. For example, process $x(z).\overline{y}\langle z\rangle.0$ means the sequences: Receive *z* through *x*; Send *z* through *y*; turn to inaction.

On the basis of behavior semantics for component composition model and combining with action prefix in π-calculus, modeling for request component, server component and the system composed of them is shown as follows using formal semantics.

Definition 9 (Request Component) Request component is composed of a group of request port and it is defined as follows:

$$R_i(r_1,...,r_m) \stackrel{\mathrm{def}}{=} \mathrm{T}_{REQ}\ \overline{r_C^1}\langle r_I^1\rangle.!\left(\mathrm{T}_{INV}\ \overline{r_I^1}\langle a^1, r_R^1\rangle.\mathrm{T}_{RES}\ r_R^1(y^1).0\right) \mid ... \mid \mathrm{T}_{REQ}\ \overline{r_C^m}\langle r_I^m\rangle.!\left(\mathrm{T}_{INV}\ \overline{r_I^m}\langle a^m, r_R^m\rangle.\mathrm{T}_{RES}\ r_R^m(y^m).0\right)$$

Based on the definition of action prefix, request $\mathrm{T}_{REQ}\ \overline{r_C^i}\langle r_I^i\rangle$ of the port must be satisfied before interaction other random ports. Request of one port can be used repeatedly once interaction channel is built on this port (refer the part after !). Since request component should work as a whole, request from all ports must be satisfied. Here parallel composition (|) among all ports is used.

Definition 10 (Server Component) Server component is composed of a group of server port and it is defined as follows:

$$P(p_1,...,p_n) \stackrel{\mathrm{def}}{=} !\Big(\mathrm{T}_{PRO}\ p_C^1(p_I^1).!\left(\mathrm{T}_{EXE}\ p_I^1(y^1, p_R^1).\mathrm{T}_{REP}\ \overline{p_R^1}\langle b^1\rangle.0\right) + ... + \mathrm{T}_{PRO}\ p_C^n(p_I^n).!\left(\mathrm{T}_{EXE}\ p_I^n(y^n, p_R^n).\mathrm{T}_{REP}\ \overline{p_R^n}\langle b^n\rangle.0\right)\Big)$$

All ports from server component can be duplicated (refer to first ! in the expression) to handle multiple requests simultaneously. Action semantics of each port is similar to that of request component. Since there is no need for server component to involve in port interaction among all ports, selective composition (+) is used between ports.

Definition 11 (Composed System) System is composed of request and server component after parallel composition:

$$ComposedSystem \stackrel{\mathrm{def}}{=} P(p_1,...,p_n) \mid R_1\left(r_{1_1},...,r_{m_1}\right) \mid ... \mid R_j\left(r_{1_j},...,r_{m_j}\right)$$

Definition 12 (Component) A component is not only the request component but also request component, i.e. it can both request and serve:

$$!\Big(\ !\big(\ \mathrm{T}_{INV}\ \overline{r_I^1}\langle a^1, r_R^1\rangle.\mathrm{T}_{RES}\ r_R^1(y^1).0 + ... + \mathrm{T}_{INV}\ \overline{r_I^m}\langle a^m, r_R^m\rangle.\mathrm{T}_{RES}\ r_R^m(y^m).0\ \big) + P(p_1,...,p_n)\ \Big)$$

Before providing any service, all requests must be satisfied, i.e. interaction channel must be established.

When server port is invoked, it will possibly trigger request port of this component to invoke service of other components. Then it will accomplish its own service.

V. COMPONENT REPLACEMENT AND EVOLUTION

During maintenance and evolution of software system, specification and realization of component may change; new upgrade version of component may be available; component may be replaced by other more competitive component [10]. Through this process, new component must be analyzed to check if it is consistent with current system environment and influences overall behavior of system. In addition, some components have to be replaced followed by change of system environment and application of new technology. This will influence overall behavior of system and lead to inconsistence. Therefore, influence of replacement to other components must be analyzed and adjustment of other components may be made to realize dynamic evolution of system.

In π calculus, 0 represents inaction process without executing any action. It is called degradation 0 when it chooses to compose left or right element of $P_1 + P_2$, otherwise it is called non-degradation 0. For example, first 0 in $0 + x(y).0$ is degradation while second one is non-degradation.

Definition 13 (Context): A context is obtained when vacancy $[.]$ replaces non-degradation 0 in process formula.

The difference between context and process is that context uses vacancy $[.]$ to replace non-degradation 0 in process. When process *P* replaces vacancy $[.]$ in context, it is denoted as $C[P]$. For example, $C_0 = \overline{z}\langle w\rangle.[.] + x(y).0$ is a context, while $C_0[z(b)] = \overline{z}\langle w\rangle.z(b) + x(y).0$ is obtained when process $z(b)$ replaces vacancy.

Each component is a process formula from the view of formal model of component. Based on definition of well-typedness, correct behavior can be shown in composed system only when process formula in composition has well-typedness, i.e. system is consistent. So consistence of system must be guaranteed when replacing component in system:

Definition 14(Consistent Replacement): Given random context *C*, when $\Gamma \vdash C[P]$ implicates $\Gamma \vdash C[P']$, component *P* can be consistent replaced by component *P'*.

A. Static Replacement

Since component uses ports to request and provide services, component replacement means ports replacement. Interaction between components is determined by contract type of port and interaction channel type between components. If component replacement only influences contract type, then interaction channel need not to be rebuilt and static analysis can be done towards contract to judge consistence of this replacement. Therefore, this component replacement is called static replacement.

Port replacement and then component replacement are discussed as follows.

Proposition 1(Consistent Replacement of Service Request Port) If $T_p(r_C) = T_p(r_C') \wedge T_c(r_C) \le T_c(r_C')$, then request port $r_C :_p T_{REQ}$ can be consistently replaced by $r_C' :_p T_{REQ}$.

Proof: From assumptions, contract ports r_C and r_C' have same kind of port type while channel type of r_C is the subtype of that of r_C', i.e. *r* is the refinement of *r'*. So *r'* has stronger pre-condition and weaker post-condition. From Lemma 3, if $\Gamma \vdash \overline{r_C}\langle r_I\rangle$, and $T_c(r_C) \le T_c(r_C')$, then $\Gamma \vdash \overline{r_C'}\langle r_I\rangle$. Since no new interaction channel needs to be rebuilt in static replacement, i.e. $T_c(r_I') \le T_c(r_I)$ is satisfied between interaction channel, then we have $\Gamma \vdash \overline{r_C'}\langle r_I'\rangle$, therefore well-typedness is preserved, i.e. the replacement of r_C' to r_C is consistent replacement.

For port replacement, its replacement environment is the component where port resides. But for component replacement, its replacement environment must include other interaction components, i.e. system composed of components. So component providing service must be included when discussing replacement of request component.

Proposition 2(Consistent Replacement of Service Request Component) let in component composition $R \mid P$, service *p* provided by *P* and service *r* requested by *R* are linked together, service *r'* provided by component *R'* is a replacement of *r*, if $T_c(p_C) \le T_c(r_C')$, then the replacement of component *R* by request component *R'* in component composition $R \mid P$, i.e. $\Gamma \vdash R \mid P \rightarrow \Gamma \vdash R' \mid P$.

Proof: Known from service *p* and *r* are linked together, $T_c(p_C) \le T_c(r_C)$ i.e. service provided by *p* satisfies service requested by *r*. Also because $T_c(p_C) \le T_c(r_C')$, so service provided by *p* satisfies service requested by *r'*. This means that replacement of *R* by *R'* will not influence composition, so this replacement maintains well-typedness and is consistent replacement.

Replacement of provide port and component is similar to that of request port and component:

Proposition 3(Consistent Replacement of Service Provide Port) if $T_p(p_C) = T_p(p_C') \wedge T_c(p_C') \le T_c(p_C)$, then provide port $p_C :_c T_{PRO}$ can be replaced consistently by $p_C' :_c T_{PRO}$.

Proof: Similar to the proof of Proposition 1.

Proposition 4(Consistent Replacement of Service Provide Component) let in component composition $R \mid P$, service *p* provided by *P* and service *r* requested by *R* are linked together, service *r′* provided by component *R′* is a replacement of *r*, if $\mathrm{T}_c(p_C{}') \le \mathrm{T}_c(r_C)$, then the replacement of component *P* by provide component *P′*I n component composition $R \mid P$, i.e. $\Gamma \vdash R \mid P \rightarrow \Gamma \vdash R \mid P'$.

Proof: Similar to the proof of Proposition 2.

B. Dynamic Replacement

Replacement of component occurred during system running is called dynamic replacement, which will change composition environment & lead reestablishment of interaction channel between components. There's a run-time environment $\mathrm{T}_{RTE}\ e(p:t_p)$ in each port to record environment information during interaction. Its semantics is denoted as $\Gamma\left[\mathrm{T}_{RTE}\ e(p:t_p)\right]$ in typing environment Γ. Dynamic replacement can be achieved by changing run-time environment of ports dynamically and reestablishment of interaction channel. For example, execution of run-time environment $\mathrm{T}_{RTE}\ e\left(r_C:\mathrm{T}_{CTR}\left(\mathrm{T}_{SIG}{}',\mathrm{PRE}',\mathrm{POST}'\right)\right)$ will change binding contract $r_C{:}\mathrm{T}_{CTR}\left(\mathrm{T}_{SIG},\mathrm{PRE},\mathrm{POST}\right)$ in typing environment Γ.

Run-time environment is introduced on the basis of formal model of component, contract of request and provide component becomes to:

$$R_i \overset{\text{def}}{=} !\left(\mathrm{T}_{RTE}\ e(r:t_r).\overline{R_i}\langle r\rangle\right) \qquad P \overset{\text{def}}{=} !\left(\mathrm{T}_{RTE}\ e\left(p:t_p\right).P(p)\right)$$

Contract of request component *R* is dynamically replaced by *R′*

$$R \overset{\text{def}}{=} !\left(\mathrm{T}_{RTE}\ e\left(r_C:t_{r_C}\right).\overline{R'}\langle r_C\rangle\right)$$

$$R' \overset{\text{def}}{=} (\ \mathrm{T}_{RTE}\ e\left(r_C:t_{r_C}\right).\overline{R'}\langle r_C\rangle + \mathrm{T}_{REQ}\ \overline{r_C}\langle r_I\rangle.(\ \mathrm{T}_{RTE}\ e\left(r_C:t_{r_C}\right).\overline{R'}\langle r_C\rangle + !\left(\mathrm{T}_{INV}\ \overline{r_I}\langle a_1,\ldots,a_s,r_R\rangle.(\ \mathrm{T}_{RTE}\ e\left(r_C:t_{r_C}\right).\overline{R'}\langle r_C\rangle + \mathrm{T}_{RES}\ r_R(y).0)\ )\ )\ )$$

It represents that execution of $\mathrm{T}_{RTE}\ e\left(r_C:t_{r_C}\right).\overline{R'}\langle r_C\rangle$ will lead the reestablishment of interaction channel (see $\mathrm{T}_{REQ}\ \overline{r_C}\langle r_I\rangle$), or re-requesting service when interaction channel remains unchanged (see $\mathrm{T}_{INV}\ \overline{r_I}\langle a_1,\ldots,a_s,r_R\rangle$), or re-receiving service result (see $\mathrm{T}_{RES}\ r_R(y)$).

Dynamic replacement transition rules of request and provide ports can be obtained based on Proposition 1& 3:

Rule 12 (Dynamic Replacement Transition Rules for Request Port) Dynamic replacement of request port $r_C:t_C$ by $r_C{}':t_C{}'$ is achieved by executing run-time environment $\mathrm{T}_{RTE}\ e\left(r_C{}':t_C{}'\right)$:

[REPL$_{\text{REQ-PORT}}$]

$$\frac{\Gamma \vdash r_C :_c t_C \quad \Gamma \vdash r_C :_p \mathrm{T}_{REQ}}{\Gamma\left[\mathrm{T}_{RTE}\ e\left(r_C{}':t_C{}'\right)\right] \vdash r_C{}' :_p t_C{}'}\langle\ t_C \le t_C{}'$$

Rule 13 (Dynamic Replacement Transition Rules for Provide Port) Dynamic replacement of request port $p_C{}':t_C{}'$ by $p_C:t_C$ is achieved by executing run-time environment $\mathrm{T}_{RTE}\ e\left(p_C{}':t_C{}'\right)$:

[REPL$_{\text{PRO-PORT}}$]

$$\frac{\Gamma \vdash p_C :_c t_C \quad \Gamma \vdash p_C :_p \mathrm{T}_{PRO}}{\Gamma\left[\mathrm{T}_{RTE}\ e\left(p_C{}':t_C{}'\right)\right] \vdash p_C{}' :_p t_C{}'}\langle\ t_C{}' \le t_C$$

From the definition of consistent replacement, replacement based on two above rules is consistent replacement.

Proposition 5 (Dynamic Replacement of Ports Can Keep Consistency) Dynamic replacement of ports based on rules [REPL$_{\text{REQ-PORT}}$] and [REPL$_{\text{PRO-PORT}}$] is consistent replacement, i.e. for $\Gamma \vdash P$, *P′* replaces *P* based on any rule of Rule 12 and 13, we have $\Gamma \vdash P'$.

Proof: Similar to the proof of Proposition 1 and 3.□

Dynamic replacement rules of component can be similarly obtained on the basis of Proposition 2 and 4.

C. Non-consistent Replacement

Following the usage of new technology and change of environment, some components have to be replaced. This replacement will influence overall behavior of system and non-consistency will appear in system. At this time, influence of replaced component to other components in system must be analyzed; then corresponding adjustment can be made to other components to realize dynamic evolution of the system.

Since components interact through the links among them, dependency among components must be analyzed based on network composed of linked components, in order to analyze influence of non-consistent replacement to the system. As we know from formal model of component, provide port may depend on request port of the port itself, while request port may also depend on provide port of other component. This dependency has transitivity. Therefore when one component is replaced, all components influenced by this replacement can be found by calculating dependency closure among ports. We should also notice that, change of provide port may still satisfy service request, so no further influence over request component.

Flow graph [11] in π calculus is used to describe space structure of linked process. It can be used to represent network composed of linked components. In the meantime, dependency graph can be obtained by considering inner dependency relation between ports of same component.

Definition 15 (Flow Graph) Nodes in flow graph are composed of ports r_C, r_I, r_R,, p_C, p_I, p_R, directed edges are composed of links (r_C, p_C), (r_I, p_I), (r_R, p_R). The direction of directed edge (r, p) is from *r* to *p*, represents that r depends on *p*.

Definition 16 (Dependency Graph) Dependency graph is the graph that is obtained by extending flow graph using inner dependency (p, r) of component.

From the definition of dependency graph, there are two kinds of edges: (r, p) represents dependency and describes sub-typing relation between components; (p, r) represents inner dependency of component and describes dependency of provide port to request port of same component.

If dependency graph after component replacement still has well-typedness, this graph is called consistent dependency graph:

Definition 17 (Consistent Dependency Graph) 如 If all edges (r, p) in dependency graph satisfies $\mathrm{T}_c(p) \le \mathrm{T}_c(r) \wedge \mathrm{T}_p(r) \Leftrightarrow \mathrm{T}_p(p)$, then this graph is consistent.

If non-consistency occurs after replacement, a certain edge in dependency will not keep well-typedness. The set of all influenced edges will be obtained by calculating dependency closure of this edge.

Definition 18 (Influence Set of Replacement) If a certain edge in dependency can not keep well-typedness, all edges influenced by this edge is the set $\{(p_i, p_j) \mid i \ge 2, j \ge 1, p_i \text{ depends on } p_j\}$.

After influence set is obtained, dynamic evolution will be realized by replacing influenced components or reestablishing channel.

VI. Related Research

In software integrated environment "Qingniao III system" [12], systematic and thoroughly research has been done on component composition. They proposed component composition technology based on system structure and described component service using syntax information on the basis of function name and parameter lists. Architecture describing language ABC (architecture based composition) /ADL is proposed too. Reference [9] proposed component composition language $\mathrm{P}_{\mathrm{ICCOLA}}$ on the basis of π calculus. $\mathrm{P}_{\mathrm{ICCOLA}}$ can support the description of different types of components and therefore it can support different kinds of component composition. It is based on modeling over interacting objects using process. The key concepts are glue code used in component composition and matching as well as script language used in describing glue code. Reference [7] expressed $\mathrm{P}_{\mathrm{ICCOLA}}$ more thoroughly. On this base, reference [8] proposed a framework of component composition and replacement based on π calculus.

Reference[2] emphasized on important roles played by contract in system correctness during the composition of the object-oriented software. By introducing specific pre-condition and post-condition in programming language, it expressed formal semantics of software and established foundation of correctness of software.

Following the development of Web applications, Web Services on the basis of component composition became research focus recently. Reference [13] combined research on Semantic Web and formal component model and discussed component contract, composition and reasoning based on web. In the meantime, Szyperski of Microsoft research center did detailed research on dynamic upgrading and expansion, composition reasoning, as well as component contract [14]. His research is based on component system structure such as .NET/COM+ and etc. He also proposed component maturity model and AsmL.

VII. Conclusion

Component composition and replacement are key technology and research focus in CBSE. Component contract semantics, behavior semantics and operational semantics are proposed in this paper. On the basis of above, formal type framework is proposed by referencing typing system in process algebra π-calculus and method in process structure. Then formal typing framework is forwarded by combining behavior semantics of process pattern in component composition and replacement. We analyze port which is important entity of composition thoroughly by using this typing framework. Port type and channel type of each port as well as different roles played by different ports in the process of component composition can be differentiated.

In the meantime, rules for composition and replacement are given on the basis of operational semantics as well as typing rules in π calculus. Rigorous analysis and reasoning can be done to correctness of composition and influence range of replacement on the basis of above rules. Error during execution is prevented to strengthen robustness and maintainability of target system.

References

[1] Yang FQ, Mei H, Li KQ. Software reuse and software component technology. Acta Electronica Sinca, 1999, 27(2):68~75 (in Chinese with English abstract).

[2] Meyer B. Object-Oriented Software Construction. Second edition. Santa Barbara: Prentice Hall Professional Technical Reference.1997.

[3] Ren HM, Qian LQ. Research on component composition and its formal reasoning. Journal of Software, 2003, 14(6):1066~1074(in Chinese with English abstract).

[4] Pahl C. A formal composition and interaction model for a web component platform. Proc. of ICALP Workshop on formal methods & component interaction. Elsevier Electronic Notes in Theoretical Computer Science, 2002, 66(4).

[5] Sangiorgi D, Walker D. The π-Calculus – A theory of Mobile Processes. Cambridge University Press, 2001.

[6] Kung-Kiu L A formal approach to software component spec. Proc. of OOPSLA Workshop on specif. & verif. of component-based systems, Iowa, 2001.88~96

[7] Lumpe M. A π-calculus based approach for software composition [Ph.D. thesis]. Bern: Universität Bern, Institut für Informatic und angewandte Mathematic, 1999.

[8] Pahl C. A Pi-calculus based framework for the composition and replacement of components[C]. Proc. of OOPSLA Workshop on specification and verification of component-based systems, Iowa, 2001.97~106.

[9] Lumpe M, Achermamm F, Nierstrasz O. A formal language for composition. In: Leavens GT, Sitaraman M,

eds. Foundations of Component-based Systems. Cambridge University Press. 2000,69~90.

[10] Vigder M, Dean J. Building maintainable COTS-based systems. In: Proceedings of the International Conference on Software Maintenance, ICSE98. IEEE Computer Society Press, 1998.132~138.

[11] Milner R. Communicating and Mobile Systems: the π - Calculus. Cambridge University Press. 1999.

[12] Yang FQ, Mei H, Li KQ, Yuan WH, Wu Q. An introduction to JB3 system supporting component reuse. Computer Science, 1999, 26(5):50~55.

[13] Pahl C. Ontology-based description and reasoning for component-based development on the web. ESEC/FSE Workshop on Specification and Verification of Component-based Systems SAVCBS'03.Helsink, Finland. 2003. 84~87.

[14] Szyperski C. Component Technology – what, where, and How? In: Proceedings of the 25th International Conference on Software Engineering (ICSE ' 03). IEEE Computer Society Press, 2003. 684~693.