Analysis and Verification of Component Behavior Equivalence for ScudWare Middleware in Ubiquitous Computing Environments

Qing Wu, Chunbo Zhao
College of Computer Science, Hangzhou Dianzi University
Email: wuqing@hdu.edu.cn
Ying Li
Zhejiang University
College of Computer Science
Email: cnliying@zju.edu.cn

Abstract— **In ubiquitous computing environments, the software component model with semantic information and behavior adaptation to satisfy various resources constraints and component interdependence is needed. It is an important issue to analyze the behavioral equivalence of components when studying the dynamic replacement and recombination of them. However it is difficult to check the equivalence of behaviors rapidly and precisely. In order to improve the precision of judging, and guarantee the normality and stability of system after replacing and recombining components for adaptation in system, the paper uses and extends the theories of equivalence analysis in π calculus, then puts forward some formalizations. After that we make some examples in detail to model the behaviors of components based on higher-order typed π calculus and analyze the equivalence of them. At last, the mobility workbench is used to make verification of this equivalence.**

Index Terms— **adaptive middleware; semantic component; component behavior equivalence.**

I. Introduction

Driven by the expansion of the network and the desire for mobility, today's computations are becoming more ubiquitous and embedded [1]. It provides more facilities and comfort for our life. Ubiquitous computing aims at fusing physical world and information space naturally and seamlessly, which demands plenty of computation resources for performance requirements. It must require considering interdependences of functional aspects. However, the computation resources in ubiquitous and embedded environments are limited such as CPU computation capabilities, network bandwidth, and memory size. As a result, it sometime cannot provide enough resources to execute some applications successfully. In addition, changes of the heterogeneous contexts including people, computing devices and environments are ubiquitous, demanding that distributed software be able to adapt to these changes. Therefore, it results in many problems in software design and development. We think software adaptation is the key issue of the systems to meet the different computing environments.

Software adaptation may require recomposition of functional aspects, which realize the imperative behavior of an application, and nonfunctional aspects, such as QoS, fault tolerance and security [2]. Specially, it attracts much attention on making research on component-based software architecture, in which replacing and recombinant of component has become one of the most attractive topics to realize software adaptation. In order to ensure the stability and reliability of the whole system, we have to analyze the behavioral equivalence of new component and replaced component.

Component behavior equivalence requires not merely interface matching, but consistency of function behavior. Some researchers neglect the function behavior and this leads to the one-sidedness of research results. Some others have used interface automata to model component behavior flow [3], [4], those behaviors having same action sequence is considered as equal. However, when one flow has more then one action sequence, the respectively equivalence of each sequence doesn't mean the equivalence of the whole flow. Thus the accuracy isn't high enough. Besides, process algebra bisimulation theory [5], [6] has also been applied to analyze the behavior equivalence. Nevertheless, no standard has been made for analyzing the component interactive behavior equivalence [7]–[9], meanwhile the one-sidedness and inaccuracy still exist. Aiming at these problems, this paper extends the bisimulation theory of π calculus, uses higher-order typed π calculus to formalized model and analyze equivalence, then apply a tool of mobility workbench to verify the results.

The remainder of the paper is organized as follows. First, we present a ScudWare middleare platform in section 2. Section 3 describes how to use higher-order typed π calculus to formalized express semantic component model and its dynamic behaviors. In section 4, we firstly propose some theories on behavioral equivalence and the

This paper is based on "A Semantic Component Model for Adaptive Middleware in Ubiquitous Computing Environments," by Wu Qing, and Li Ying, which appeared in the Proceedings of the Second International Workshop on Computer Science and Engineering, 28-30 Oct.2009,Qingdao, China.

This work was supported by National Natural Science Foundation of China under Grant No. 60703088.

doi:10.4304/jsw.5.8.883-890

operation method of MWB [10], then give a case study to verity our model and method. Next, some related work is stated in section 5. Finally, we draw a conclusion and give further research work in section 6.

II. ScudWare Middleware Platform

We have developed a semantic and adaptive middleware platform called *ScudWare* [11], which is based on semantic information and conformed to a lightweight CCM (CORBA component model) specification [12].

In this section, we give the ScudWare architecture firstly. And then we introduce a smart CAA space in ubiquitous computing environments.

A. ScudWare Middleware Architecture

ScudWare architecture consists of five parts defined as *SCUDW* = (*ACE, ETAO, SCUDCCM, SVA*). *ACE* denotes the adaptive communication environment [13], providing high-performance and real-time communications. *ACE* uses inter-process communication, event demultiplexing, explicit dynamic linking, and concurrency. In addition, *ACE* automates system configuration and reconfiguration by dynamically linking services into applications at run-time and executing these services in one or more processes or threads. *ETAO* extends ACE ORB [14] and is designed using the best software practices and patterns on ACE in order to automate the delivery of high-performance and real-time QoS to distributed applications. *ETAO* includes a set of services such as the persistence service and transaction service. In addition, we have developed an adaptive resource management service, a context service and a notification service. Specially, the context service is based on semantic information. *SCUDCCM* is conformed to CCM specification and consists of adaptive component package, assembly, deployment, and allocation at design-time. Besides, it comprises component migration, replacement, updating, and variation at run-time. In addition, the top layer is *SVA* that denotes semantic virtual agent [15]. *SVA* aims at dealing with application tasks. Each *sva* presents one service composition comprising a number of meta objects. During the cooperations of *SVA*, the SIP(Semantic Interface Protocol) set is used including *sva* discovery, join, lease, and self-updating protocols.

B. Smart Vehicle Space

Vehicles currently have played a very important role for improving our living. People hope to have a comfortable, convenient, and safe environment in the vehicle. This has been a greatly significant requirement in the ITS (intelligent transportation system) community [16]–[18]. An appealing way, also a very challenging task, is to build a ubiquitous computing environment in the vehicles.

Physical world and information spaces fuse naturally and spontaneously in smart spaces. All entities in smart spaces self-adjust adaptively in terms of the changes of the users, environments and devices for communication and cooperation.

In recent years, many researchers and engineers have weaved embedded, AI and biological authentication technologies into vehicles for developing and deploying ITS subsystems and services. The drive capability, dependability, comfort and convenience of the vehicle are greatly improved. After persons get into the smart vehicle space, they will find many smart and intelligent devices around them. They will communicate with these tools naturally and friendly to get more useful information and services. It will form a smart and harmonious vehicle space. To achieve adaptation and transparentness of co-operations and integrations among persons, devices and environments in ITS, the intersection between smart space and ITS(smart vehicle space) plays an important role.

From the technical view, the smart vehicle space has four parts and is defined as

$SVS = (CA, CR, AC, CP)$

CA: the context acquisition system. It aims at sensing status changes of people, devices and environments in the vehicle, including cameras, sound receivers, and other sensors.

CR: the context repository reasoning system. *CR*=(*context, ontology, domain, inference*) uses the correlative contexts and application domain ontologies to make manipulating strategy for purpose of adaptation.

AC: the auto controlling system. It consists of the steering, communication, entertainment, navigation and security subsystem.

CP is the centralized processing system. It is the kernel of the smart vehicle space, which makes *CA, CR, AC* collaborate effectively.

III. Semantic Component Model for ScudWare Middlware

In ScudWare architecture, the semantic components are essential software entities. These components implement some application logic and can execute special functions when they are instantiated. The structure properties, function behaviors, and inner adaptive behaviors are three important parts of the semantic components. Specially, the component inner adaptive behaviors can change component resources consumption states to get satisfying execution effect required from other components in the ScudWare middleware system. The system components can provide runtime environments infrastructure such as context-aware information and adaptive behaviors managements for semantic components. As a result, the semantic and system components are executors of functional and non-functional behaviors of ScudWare middleware.

In the following, we present a semantic component formalization and the semantic component dynamic behaviors modeling.

A. Semantic Component Formalization

We use ScudADL [19] to define the semantic component, describing its structure character and dynamic behavior.

An semantic component $A_C ::= Name|Ontology| < \widetilde{Cap} > | < \widetilde{Port} > | < \widetilde{RI} > | < \widetilde{ExeModel} > | < \widetilde{FuncBeha} > | < \widetilde{AdapBeha} >$, $\widetilde{AdapBeha} ::= < \widetilde{IAdapBeha} > | < \widetilde{OAdapBeha} >$

1) $Ontology$ is a repository of components. Component ontology provides common and sharing conceptual understanding of specific domain for functions and behavior of components.

2) $< \widetilde{Cap} >$ denotes a semantic description of component's capabilities, including a set of computation functions.

3) $< \widetilde{Port} >$ denotes a set of input interfaces provided by other components, and a set of interfaces exporting for other components use.

4) $< \widetilde{RI} >$ is component resource interface, denoting a set of required resources consumptions value (e.g. computation platform type, CPU computation, network communication bandwidth, and memory size).

a) *CPU Computation Consumption*: $RC_{cc} : \forall c \in A_c \cdot \exists v \in Q^+ \cdot (RC_{cc} \rightarrow v)$ defines the CPU computation resource consumption by component c. Q^+ is a set of non-negative real numbers.

b) *Communication Consumption*: $RC_{cm} : \forall c \in A_c \cdot \exists v \in Q^+ \cdot (\sum RC_{cm} \rightarrow v)$ defines communication resource consumption by component c.

c) *Memory Consumption*: $RC_{mm} : \forall c \in A_c \cdot \exists v \in Q^+ \cdot (RC_{mm} \rightarrow v)$ defines memory resource consumption by component c.

5) $< \widetilde{ExeModel} >::== (< \widetilde{Res}, \widetilde{ExeQua} >)$. On the condition of different component resource consumption, it will provide different execution effect in the whole middleware system. For example, $((RC^i_{cc}, RC^j_{cm}, RC^k_{mm}), EQ^\omega_{ac})$ denotes that if one component consume RC^i_{cc} cpu computation resource, RC^j_{cm} communication resource, and RC^k_{mm} memory resource, it will provide EQ^ω_{ac} execution quality.

6) $< \widetilde{FuncBeha} >::==< \widetilde{IO} >< \widetilde{FuncBeha} > | < \widetilde{Condition} >< \widetilde{FuncBeha} > | < \widetilde{Choose} >< \widetilde{FuncBeha} > |unobservable|inaction$. The component function behaviors include a) input and output operations via channel and resource channel, b) condition operation (if ... then ...), corresponding to $[X = Y]P$ in higher-order typed π calculus, c) choose operation, corresponding to $P|Q$, d) unobservable operation, corresponding to τ, e) inaction operation, corresponding to 0.

a) $\widetilde{IO} ::= < \widetilde{SMessage} > | < \widetilde{RMessage} > | < \widetilde{SExeModel} > | < \widetilde{RExeModel} >$. Input operation consists of send or receive message via channel, and send or receive execution model via resource channel.

b) $SMessage ::=$ via Channel $(< \widetilde{Port} >)$ Send message

c) $RMessage ::=$ via Channel$(< \widetilde{Port} >)$ Receive message

d) $SExeModel ::=$ via ResChannel$(< \widetilde{RI} >)$ Send ExeModel

e) $RExeModel ::=$ via ResChannel$(< \widetilde{RI} >)$ Receive ExeModel

7) $IAdapBeha ::= < \widetilde{ChangeExeModel} > .IAdapBeha$. Inner adaptive behavior is to change the component execution model in terms of variable computing environment or application requirements. It can change the component resources consumption and get a new execution quality. The execution model is from $((RC^i_{cc}, RC^j_{cm}, RC^k_{mm}), EQ^\omega_{ac})$ to $((RC^p_{cc}, RC^q_{cm}, RC^r_{mm}), EQ^\mu_{ac})$.

8) $OAdapBeha ::= AddAc.OAdapBeha | RemoveAc.OAdapBeha | UpdateAc.OAdapBeha | ReplaceAc.OAdapBeha | inaction$. In outer adaptive behaviors, a) $AddAc$ behavior denotes add a new component into the system dynamically for a new functionality, b) $RemoveAc$ behavior denotes remove a old component from the system, which is not necessary, c) $UpdateAc$ behavior denotes updating component functionality to a new version, d) $ReplaceAc$ behavior denotes replacing one component with another component, continuing to conduct the next operations between other components.

a) $AddAc$ behavior. When a new component is added into the system, the component instance, its port, resource interface, and channel will be built according to application requirements. The connector and resource connector in the system will build a new link to this new component and adjust the routing behavior.

b) $RemoveAc$ behavior. When an old component is not needed, it will be removed by the system. The component instance, its port, resource interface, and channel will be destroyed in terms of removal rules. The connector and resource connector in the system will delete the links of this component.

c) $UpdateAc$ behavior. A^i_c can update to A^j_c after executing $UpdateAc$ behavior. If the functional behavior of A^i_c corresponds to process P, and the functional behavior of A^j_c corresponds to process Q, then P and Q are strongly bisimilar, which is a strong equivalents relation ($P \sim Q$).

d) $ReplaceAc$ behavior. A^i_c can be replaced with A^j_c in the system after executing $ReplaceAc$ behavior. If the functional behavior of A^i_c corresponds to process P, and the functional behavior of A^j_c corresponds to process Q, then P and Q are weakly bisimilar, which is a week equivalence relation ($P \approx Q$).

B. Semantic Component Dynamic Behaviors Modeling

The semantic component behaviors have dynamic and concurrent characters. In addition, the component interacts with others through its service request ports and service supply ports, whose interaction is mainly embodied in messages transfer. Similarly, the processes of π calculus transfer messages with others through its channels. Thus we can map the component ports to the π calculus process channels. And the transceivers of messages by components correspond to the transceivers of messages by π calculus process.

According to the different transfer forms of message, the component atomic behaviors can be divided into three

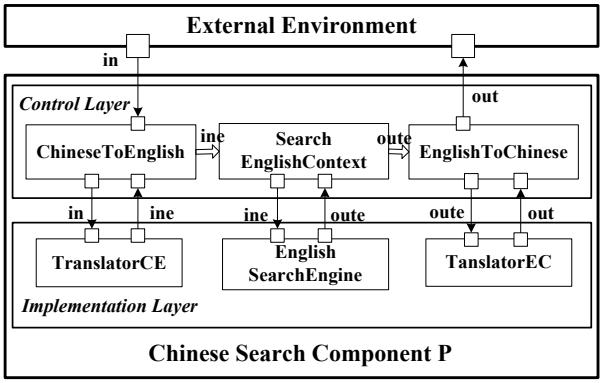

Figure 1. Behavior View of Chinese Search Component P

kinds. The first is *send only(S)*. The second is *receive only(R)*. And the third are *send before receive(SR)* and *receive before send(RS)*.

The set of all communication ports is defined as GT. gt_i is one communication port. $GT = \{gt_1, gt_2, ..., gt_n\}$

The set of all input messages is defined as M_{in}. i_u is one input message. $M_{in} = \{i_1, i_2, ..., i_u\}$

The set of all output messages is defined as M_{out}. o_v is one output message. $M_{out} = \{o_1, o_2, ..., o_v\}$

$gt_n?(i_u)$ denotes one component receiving message i_u through port gt_n, while $gt_n!(o_v)$ denotes the component sending message o_v through port gt_n. The component atomic behaviors of $send$, $receive$, $send, receive$ and $receive, send$ through ports are defined as follows.

$P_{send} = gt_m!(o_j, o_{j+1}, ..., o_{k-1}, o_k) \cdot 0$

$P_{receive} = gt_m?(i_j, i_{j+1}, ..., i_{k-1}, i_k) \cdot 0$

$P_{send,receive} = gt_m!(o_j, ..., o_k) \cdot gt_m?(i_j, ..., i_k) \cdot 0$

$P_{receive,send} = gt_m?(i_j, ..., i_k) \cdot gt_m!(o_j, ..., o_k) \cdot 0$

For instance, a network-based search engine in ubiquitous computing environments have one component called Chinese search component P, which has three subcomponents those are $TranslaotrCE$, $EnglishSearchEngine$ and $TranslatorEC$. The input and output channels of P are named as $ptIN$ and $ptOUT$, while the channels of $TranslatorCE$, $EnglishSearchEngine$ and $TranslatorEC$ are called $ptEC$, $ptSEC$ and $ptCE$. The Chinese search component dynamic behaviors are illustrated in figure 2.

If P_{CSE} represents the dynamic behaviors of P, then it can be described as follows.

$P_{CSE} = pt_{IN}?(in).P1$

$P1 = (vf_{INE}(pt_{CE}!(in).pt_{CE}?ine.f_{INE}!(ine).0$
$||f_{INE}?(ine).P2)$

$P2 = (vf_{OUTE}(pt_{SEC}!(ine).pt_{SEC}?(oute).f_{OUTE}$
$!(oute).0)||f_{OUTE}?(oute).P3$

$P3 = pt_{EC}!(oute).pt_{EC}?(out).pt_{OUT}!(out).0$

The component evolution is shown as below.

$P_{CSE} \xrightarrow{pt_{IN}?(in)} P1$

$\xrightarrow{pt_{CE}!(in)} (vf_{InE})(pt_{CE}?(ine).f_{INE}!(ine).0$
$||f_{INE}?(ine).P2)$

$\xrightarrow{pt_{CE}!(ine)} (vf_{INE}(f_{INE}!(ine).0||f_{INE}?(ine).P2))$

$\xrightarrow{\tau(f_{INE})} P2$

$\xrightarrow{pt_{SEC}!(ine)} (vf_{OUTE})(pt_{SEC}?(oute).f_{OUTE}!(oute).0$
$||f_{OUTE}?(oute).P3)$

$\xrightarrow{pt_{SEC}!(oute)} (vf_{OUTE}(f_{OUTE}!(oute).0$
$||f_{OUTE}?(oute).P3))$

$\xrightarrow{\tau(f_{OUTE})} P3$

$\xrightarrow{pt_{EC}!(oute)} pt_{EC}?(out).pt_{OUT}!(out).0$

$\xrightarrow{pt_{EC}?(out)} pt_{OUT}!(out).0$

$\xrightarrow{pt_{OUT}!(out)} 0$

IV. Analysis and Verification of Semantic Component Behaviors Equivalence

In terms of the semantic component model and its dynamic behavior formalization, we will propose the analysis and verification of semantic component behaviors equivalence. In this section, we give the component behavior equivalence based on higher-order π calculus firstly. Then the mobility workbench is introduced. Finally, we give a case study and its verification.

A. Component behavior equivalence

Here we will apply and extend the bisimulation theory of higher-order π calculus, then put forward some standards for component behaviors equivalence analysis, which are applicable in different conditions. In the following, we give some definitions of component equivalence.

1) *Component Context (CC)*. Here, we divide component context into external environment and internal environment. The former is reciprocal behaviors of inside subcomponent interactive behaviors, while the latter is reciprocal behaviors of interactions with other components. If a component P has a set of outer behaviors named as X, a set of inner behaviors named as Y, and a context CP, well then CP can be described as follows.

$C_P = \alpha_i^{-1} + C_{P'}$

$P(\xrightarrow{\tau})^* \xrightarrow{\alpha_i} (\xrightarrow{\tau})^* P'$, $\alpha_i \in X$

α_i^{-1} is an oppositional behavior of α_i. If $\alpha_i = \alpha_i!(x)$, then $\alpha_i^{-1} = \alpha_i?(x)$. And if $\alpha_i = \alpha_i?(x)$, then $\alpha_i^{-1} = \alpha_i!(x)$.

For example, component P has two subcomponents names Pa and Pb, and the interactive behaviors between P and Q is concretely described in figure 3. Well then some related interactive behaviors can be described as below.

(1) Interactive behavior between P and Q:
$pt_{PIN}?(in).pt_{POUT}!(out).0$

(2) Interactive behavior between Pa and Pb:
$pt_{PaIN}?(in).pt_{PaOuT}!(outa).0||$
$pt_{PbIN}?(outa).pt_{PbOUT}!(out).0$

(3) Internal Environment of P:
$(pt_{PaIN}!(in).\tau.pt_{PaOUT}?(outa))^*||$
$(pt_{PbIN}!(outa).\tau.pt_{PbOUT}?(out))^*$

(4) External Environment of P:
$pt_{PIN}!(in).pt_{POUT}?(out).0$

2) *Higher-order Strong Bisimulation(HSB)*. HSB ($\cong$) is the largest symmetric relation between processes such that $P \cong Q$ implys:

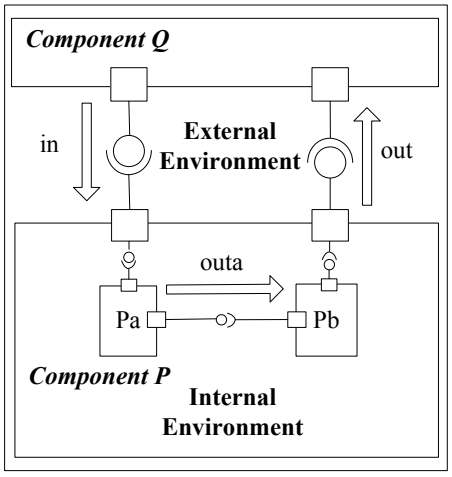

Figure 2. Interactive Behaviors of P and Q

Input	Translation
^	*restriction*
\	*abstraction*
0	*null process*
'a	*output action*
t	*internal action*
a*(nlist)*	*input prefix*
'a < *nlist* >	*outputprefix*

Figure 3. Basic Input Grammar

P::=0	*the inactive process*
a.P	*perform the action a and continue as P*
[a=b]P	*if a = b then run P*
P1|P2	*run P1 and P2 in parallel*
P1+P2	*run either as P1 or P2*
Id(*nlist*)	*run as the process named Id instantiated with the names in nlist*
(^*nlist*)P	*restrict the names in nlist*
(P)	*parentheses are used for enforcing precedence*

Figure 4. Basic Operation Semantics

(1) $\alpha \in Q$, $\alpha \in \tau, a?(x), a!(x)$, in which x is a name or process;

(2) If $Q \xrightarrow{\alpha} Q^{'}$, then $P \xrightarrow{\alpha} P^{'}$ and $P^{'} \cong Q^{'}$.

HSB requires one-to-one correspondence of action in different components behaviors, so the equivalent conditions are rigorous.

3) *Higher-order Weak Bisimulation(HWB)*. HWB (:) is a comparatively weak relation between processes such that : implys:

(1) $\alpha \in Q$, $\alpha \in \tau, a?(x), a!(x)$, in which x is a name or process;

(2) If $Q \xrightarrow{\alpha} Q^{'}$, then $P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* P^{'}$ and $P^{'} : Q^{'}$.

Much attention should be paid to the bisimulation strength difference between HSB and HWB. As HSB's over high bisimulation and HWB's regardless of internal actions, HWB is much more widely used than HSB. However, HSB is always needed in high stability and reliability system.

4) *Context Compatibility Bisimulation(CCB)*. CCB ($\diamondsuit$) is a relation between processes such that $P^{'} \diamondsuit Q^{'}$ implys:

(1) C_P is a context of P;

(2) If $P||C_P(\xrightarrow{\tau})^*.0$ then $Q||C_P(\xrightarrow{\tau})^*.0$.

Aiming at the compatibility of new component with replaced component context, CCB always is a weaker bisimulation than HWB.

B. Mobility Workbench

Mobility workbench (MWB) is the first automatic verification tool of π calculus, which can be used to model and verify the mobile concurrent system described by process calculus. This paper adopts this tool to formalized model, analyze and verify the equivalence among components. The operation semantics and commands of MWB are given as follows.

Some basic input grammar and operation semantics are shown in figure 4 and figure 5.

Some general commands are listed as below.

(1) $Agent$ can be used to create a closed process with a list of parameters.

(2) Env can show all agents defined, also it can display appointed agent when followed by the agent name.

(3) $Clear$ will delete all agents defined.

(4) $Input$ followed by a filename with double quote (*input "filename"*) will put the file content into MWB.

(5) Eq followed by two agent names (*eq agent1 agent2*) will check strong bisimulation of agents.

(6) Weq followed by two agent names (*weq agent1 agent2*) will check weak bisimulation of agents.

Except for the commands mentioned above, more introduction can be found in paper '*A Brief Introduction to Mobility Workbench*'.

C. Case Study and Verification

According to the Chinese search component P in ubiquitous computing environments mentioned above, we can create some other Chinese search components, then analyze and verify the behavioral equivalence of them.

Firstly, a Chinese search component named Q is created, which can return Chinese search result according to the input search terms. After obtaining search words, Q will choose one of the following processing methods: 1) using the Chinese search engine directly, or 2) firstly converting the Chinese search terms into English, then using the English search engine, at last converting search result back to Chinese. The behavior view of component Q is concretely described in figure 6.

Assume that Q has the input and output channels successively named as pt_{QIN} and pt_{QOUT}, while the channels of Chinese search engine, English search engine, $TranslatorEC$ and $TranslatorCE$ are successively described as pt_{QSCC}, pt_{QSEC}, pt_{QEC} and pt_{QCE}. If the process Q_{CSE} represents interactive behavior of Q, then it can be described as follows.

1) $Q_{CSE} = pt_{QIN}?(in).Q1$

2) $Q1 = (vrandom)(pt_{RANDOM}!(random).0||$

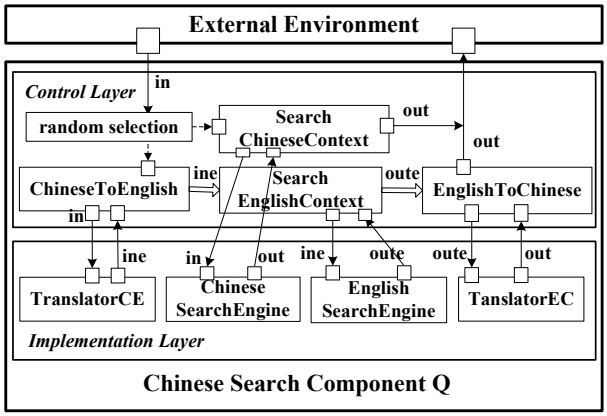

Figure 5. Behavior View of Chinese Search Component Q

```
agent Subcomponent(i,o,invalue,outvalue)=i(invalue).'o<outvalue>.0
agent Pcse(ptin,ptout,invalue,outvalue)=(^ptce,cevalue,ptsec,secvalue)
        (Subcomponent(ptin,ptce,invalue,cevalue)
        |Subcomponent(ptce,ptsec,cevalue,secvalue)
        |Subcomponent(ptsec,ptout,secvalue,outvalue))
agent Qcse(ptin,ptout,invalue,outvalue)=(^ptce,cevalue,ptsec,secvalue)
        (Subcomponent(ptin,ptce,invalue,cevalue)
        |Subcomponent(ptce,ptsec,cevalue,secvalue)
        |Subcomponent(ptsec,ptout,secvalue,outvalue)
        +Subcomponent(ptin,ptout,invalue,outvalue))
```

Figure 6. Component Behavior Description 1

$pt_{RANDOM}?(random).(Q2 + Q3)$

3) $Q2 = pt_{QSCC}!(in).pt_{QSCC}?(out).pt_{QOUT}!(out).0$

4) $Q3 = (vf_{INE}(pt_{QCE}!(in).pt_{QCE}?(ine).f_{INE}!(ine).0||$
$f_{INE}?(ine).Q31$

5) $Q31 = (vf_{OUTE})(pt_{QSEC}!(ine).pt_{QSEC}?(oute).$
$f_{OUTE}!(oute).0||f_{OUTE}?(oute).Q32)$

6) $Q32 = pt_{QEC}!(oute).pt_{QEC}?(out).pt_{QOUT}!(out).0$

The component evolution is shown as below.

1) $Q_{CSE} \xrightarrow{pt_{QIN}?(in)} Q1 \xrightarrow{\tau(pt_{RANDOm})} Q2 + Q3$

2) $Q2 \xrightarrow{pt_{QSCC}!(in)} pt_{QSCC}?(out).pt_{QOUT}!(out).0$
$\xrightarrow{pt_{QSCC}?(out)} pt_{QOUT}!(out).0 \xrightarrow{pt_{QOUT}!(out)} 0$

The evolution of process $Q3$ is similar to the process $P1$ in P_{CSE} and it's no longer repeated here.

After comparing the evolution of P_{CSE} and Q_{CSE}, it can be easily found that component behaviors of P and Q are higher-order weak bisimulation ($P : Q$). However, Q_{CSE} needs random selection($\tau(pt_{RANDOM})$) between behaviors of $pt_{QIN}?(in)$ and $pt_{QSCC}!(in)$, while P_{CSE} needn't it. Therefore, P and Q aren't higher-order strong bisimulation.

After that, we use the mobility workbench to model and verify the equivalence of P_{CSE} and Q_{CSE}. We create three agents: 1)$Subcomponent$ that represents TranslatorCE, Chinese SearchEngine, English SearchEngine and TranslatorEC, 2)P_{CSE} that represents behavior of P, 3)Q_{CSE} that represents behavior of Q. They are described in figure 7.

Then we can import these agents into MWB by using command 'input' followed by a filename. After that commands 'weq' and 'eq' are used to check the

```
MWB>input "CheckPQ.txt"
MWB>weq Pcse Qcse
The two agents are equal.
Bisimulation relation size = 7.
MWB>eq Pcse Qcse
The two agents are NOT equal.
```

Figure 7. Equivalence Verification Result 1

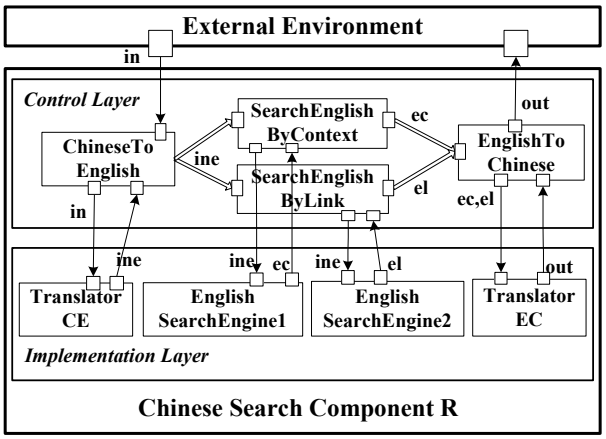

Figure 8. Behavior View of Chinese Search Component R

equivalence of P_{CSE} and Q_{CSE}. As is shown in figure 8, P_{CSE} and Q_{CSE} are higher-order weak bisimulation and the bisimulation relation size is 7.

After comparing the component behavior of P and Q, then we create another Chinese search component R, which also can return Chinese search result after receiving Chinese search terms. R has four subcomponents: 1)$TranslatorCE$ for translating Chinese into English, 2) $EnglishSearchEngine1$ for searching English by context, 3)$EnglishSearchEngine2$ for searching English by link, 4)$TranslatorEC$ for translating English into Chinese. The channels of them are successively described as pt_{RCE}, pt_{RSEBC}, pt_{RSEBL} and pt_{REC}. The interactive behavior view of R is described in figure 9.

Assume that process R_{CSE} represents interactive behaviors of R, and then it can be formalized described as follows.

1)$R_{CSE} = pt_{RIN}?(in).R1$

2)$R1 = (vf_{INEC}, f_{INEL})(pt_{RCE}!(in).pt_{RCE}?(ine).$
$(f_{INEC}!(ine)||f_{INEL}!(ine)).0||f_{INEC}?(ine).R2$
$||f_{INEL}?(ine).R3$

3)$R2 = (vf_{OUTEC}, f_{OUTEL})(pt_{RSEBC}!(ine).$
$pt_{RSEBC}?(ec).f_{OUTEC}!(ec).0||f_{OUTEC}?(ec).$
$f_{OUTEL}?(el).R4)$

4)$R3 = (vf_{OUTEC}, f_{OUTEL})(pt_{RSEBL}!(ine).$
$pt_{RSEBL}?(el).f_{OUTEL}!(el).0||f_{OUTEL}?(el).$
$f_{OUTEC}?(ec).R4)$

5)$R4 = pt_{REC}!(ec).pt_{REC}!el.pt_{REC}?(out).$
$pt_{ROUT}!(out).0$

The evolution of R_{CSE} is listed as below:

1)$R_{CSE} \xrightarrow{pt_{RIN}?(in)} R1 \xrightarrow{pt_{RCE}!(in)} (vf_{INEC}, f_{INEL})$
$(pt_{RCE}?(ine).(f_{INEC}!(ine)||f_{INEL}!(ine)).0||$

$f_{INEC}?(ine).R2||f_{INEL}?(ine).R3)$
$\overset{pt_{RCE}?(ine)}{\longrightarrow}(vf_{INEC},f_{INEL})((f_{INEC}!(ine)||$
$f_{INEL}!(ine)).0||f_{INEC}?(ine).R2||f_{INEL}?(ine).R3)$
$\overset{\tau(f_{INEC},f_{INEL})}{\longrightarrow} R2||R3$
$2)R2 \overset{pt_{RSEBC}!(ine)}{\longrightarrow}(vf_{OUTEC},f_{OUTEL})(pt_{RSEBC}?(ec).$
$f_{OUTEC}!(ec).0||f_{OUTEC}?(ec).f_{OUTEL}?(el).R4)$
$\overset{pt_{RSEBC}?(ec)}{\longrightarrow}(vf_{OUTEC})(f_{OUTEC}!(ec).0||$
$f_{OUTEC}?(ec).f_{OUTEL}?(el).R4)$
$3)R3 \overset{pt_{RSEBL}!(ine)}{\longrightarrow}(vf_{OUTEC},f_{OUTEL})(pt_{RSEBL}?(el).$
$f_{OUTEL}!(el).0||f_{OUTEL}?(el).f_{OUTEC}?(ec).R4)$
$\overset{pt_{RSEBL}?(el)}{\longrightarrow}(vf_{OUTEL})(f_{OUTEL}!(el).0||$
$f_{OUTEL}?(el).f_{OUTEC}?(ec).R4)$
$4)R2||R3 \overset{\tau(f_{OUTEL},f_{OUTEC})}{\longrightarrow} R4 \overset{\tau,pt_{REC}!(ec)}{\longrightarrow}$
$pt_{REC}!(el).pt_{REC}?(out).pt_{ROUT}!(out).0$
$\overset{pt_{REC}!(el)}{\longrightarrow} pt_{REC}?(out).pt_{ROUT}!(out).0$
$\overset{pt_{REC}?(out)}{\longrightarrow} pt_{ROUT}!(out).0$
$\overset{pt_{ROUT}!(out)}{\longrightarrow} 0$

According to the whole evolution of R_{CSE}, the concurrent processes $R2||R3$ can evolute to $f_{OUTEC}?(ec).f_{OUTEL}?(el).R4$ after a series of actions, while the equivalent state can't be reached P_{CSE}. In addition, the action of $pt_{REC}!(el)$ will be made after $pt_{REC}!(ec)$ in R_{CSE}, but it's impossible in P_{CSE} even if performing some internal actions. Thus, P_{CSE} and R_{CSE} aren't higher-order weak bisimulation.

On the basis of definition ***Component Context***, the component context of P_{CSE} can be described as follows.

$1)C_{P_{CSE}} = CP_{Input}||CP_{ChineseToEnglish}||$
$CP_{SearchEnghish}||CP_{EnglishToChinese}||CP_{Output}$
$2)CP_{Input} = pt_{IN}!(in).0$
$3)CP_{ChineseToEnglish} = (pt_{CE}?(in).\tau.pt_{CE}!(ine))^*$
$4)CP_{SearchEnghish} = (pt_{SE}?(ine).\tau.pt_{SE}!(oute))^*$
$5)CP_{EnglishToChinese} = (pt_{EC}?(oute).\tau.pt_{EC}!$
$(oute))^*$
$6)CP_{Output} = pt_{OUT}?(out)$

Judging from the interactive behaviors of R, R_{CSE} can successfully complete its behaviors in the component context of $C_{P_{CSE}}$, that is $R_{CSE}||C_{P_{CSE}}(\overset{\tau}{\longrightarrow})^*.0$. Consequently, P_{CSE} and R_{CSE} are context compatibility bisimulation ($PCSE \diamondsuit SSE$).

After that, we use the MWB tool to model and verify the equivalence of P_{CSE} and R_{CSE}. According to the concrete behaviors of composite component R and P, we create four agents: 1) Subcomponent that represents $TranslatorCE$ and $EnglishSearchEngine$, 2)$EnglishToChinese$ that represents $TranslatorEC$ in R, 3)P_{CSE} that represents the behavior of Chinese search component P, and 4)R_{CSE} that represents the behavior of Chinese search component R. They are described in figure 10 in detail.

After checking the equivalence of P_{CSE} and R_{CSE}, we find they are neither higher-order string bisimulation, nor higher-order weak bisimulation, and the result is obviously shown in figure 11.

```
agent Subcomponent(i,o,invalue,outvalue)=i(invalue).'o<outvalue>.0
agent Englishtochinese(i,o,invalue1,invalue2,outvalue)=
        i(invalue1,invalue2).'o<outvalue>.0
agent Pcse(ptin,ptout,invalue,outvalue)=
        (^ptce,cevalue,ptsec,secvalue)(Subcomponent(ptin,ptce,invalue,cevalue)
        |Subcomponent(ptce,ptsec,cevalue,secvalue)
        |Subcomponent(ptsec,ptout,secvalue,outvalue))
agent Rcse(ptin,ptout,invalue,outvalue)=
        (^ptce,ptse,cevalue,sebcvalue,seblvalue,ptec)
        (Subcomponent(ptin,ptce,invalue,cevalue)
        |(Subcomponent(ptce,ptse,cevalue,sebcvalue)
          +Subcomponent(ptce,ptse,cevalue,seblvalue))
        |Englishtochinese(ptse,ptout,sebcvalue,seblvalue,outvalue))
```

Figure 9. Component Behavior Description 2

```
MWB>input "CheckPR.txt"
MWB>eq Pcse Rcse
The two agents are NOT equal.
MWB>weq Pcse Rcse
The two agents are NOT equal.
```

Figure 10. Equivalence Verification Result 2

V. RELATED WORK

It's an important issue to analyze the behavioral equivalence of components when studying the dynamic replacement and recombination of them for software adaptation.

The project ArchWare [20] developed by Europe Unite aims to construct a evolvable system centered with architecture mode. ArchWare has proposed a dynamic architecture Description Language π-ADL [21] that is a formal language that based on higher-order π calculus, which is supporting modeling dynamic architectures, analyzing architectures and checking constraints. In addition, D-ADL [22] explicitly defines two dynamic behavior operations symbols 'new' and 'delete', which is easier to describe and comprehend dynamic behaviors. Therefore, D-ADL implements the description of architecture dynamic behavior, and provides an indirect supporting for architecture evolution.

In addition, we have acquired some achievements on how to formalized model component behavior, how to analyze the equivalence of component interactive behaviors. T.Basten [3], Zhang [4] has used interface automata to express component behavior, and it's intuitive to describe behaviors in graphics mode, but the computational complexity will increase rapidly when interactive behaviors become involuted. Xu [5] has put forward some bisimulation equivalence theories of higher-order π calculus, but no practical case based on these theories has been given. Gao [23] has used π calculus to model behavior, but they haven't verified the equivalence of different behaviors. In order to ensure the stability and high performance of the whole system after replacing or recombining components, we must analyze and ensure the equivalence of new component and replaced component. Focusing on this problem, this paper emphatically model and analyze the equivalence of behaviors among components and then use mobility workbench to verify it.

VI. Conclusion and future work

This paper firstly describes how to use higher-order π calculus to formalized model component behavior, and then we put forward some theories for equivalence analysis. At last, we make an illustrative example of Chinese search component in ubiquitous computing environments to analyze behavioral equivalence, and verification is made by using mobility workbench. Through the precise analysis and verification of equivalence of component behavior, we can ensure the behavioral consistency of new component and replaced component, also effectively guarantee the stability and normality of the whole system's dynamic adaptation after replacing and recombining of component.

Our next step work includes that 1) we will make further research on behavioral equivalence of components. 2) except for the functional behavior, the evolution behavior should be considered as well.

Acknowledgment

We thank Li Changyun for numerous discussions concerning this work, and the reviewers for their detailed comments.

References

[1] Weiser M. The Computer for the 21st Century, *Scientific American*, pp. 94-100, 1991.
[2] E.P.Kasten and P.K.McKinley. Perimorph: Run-Time Compostion and State Management for Adaptive Systems. In Proceedings of the 24th International Conference on Distributed Computing Systems Workshops. 2004.
[3] W.M.P.van der Aalst and T.Basten. Inheritance of Workflows:An Approach to Tackling Problems Related to Change[J]. Theoretical Computer Science, 270(1-2):125-203, 2002.
[4] Yan Zhang, Jun Hu, XiaoFeng Yu. Scene Drivn Component Behavior Extraction[J]. Journal of software, 2007, 18(1):50-61.
[5] Xan Xu. On the Bisimulation Theory and Axiomatization of Higher-order Process Calculi[D]. Doctor degree dissertation of ShangHai Jiao Tong University,2008.
[6] Stephanie Delaune, Steve Kremer, Mrak Ryan. Symbolic Bisimulation for the Applied Pi Calculus[J]. Foundations of Software Technology and Theoretical Computer Science, 2007.
[7] Kuang, Li. A formal analysis of behavioral equivalence for web services. IEEE Congress on Services,2008,265-268.
[8] W.M.P. Van Der Aalst, A.K. De Alves Medeiros, A.J.M.M. Weijters. Process Equivalence:Comparing Two Process Models Based on Observed Behavior[J]. Business Process Management(BPM),129-144.2006.
[9] Huimin Lin. Inference systems for observation equivalences in the -calculus[J]. Technological Sciences,1999.
[10] Mikkel Bundgaard. A Brief Introduction to Mobility Workbench(MWB). Department of Theoretical Computer Science, IT University of Copenhagen,2005.
[11] Zhaohui Wu, Qing Wu, Hong Cheng, Gang Pan, and Minde Zhao, SCUDWare: A Semantic and Adaptive Middleware Platform for Smart Vehicle Space, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 8, No. 1, pp. 121-132, 2007.
[12] http://www.omg.org/technology/documents/formal/components.htm, 2005.
[13] http://www.cs.wustl.edu/ schmidt/ACE.html, 2005.
[14] http://www.cs.wustl.edu/ schmidt/TAO.html, 2005.
[15] Qing Wu and Zhaohui Wu, Semantic and Virtual Agents in Adaptive Middleware Architecture for Smart Vehicle Space, *In proceeding of the 4th International Central and Eastern European Conference on Multi-Agent Systems*, Springer LNAI 3690, pp. 543-546, 2005.
[16] F.Y. Wang et al., Toward Intelligent Transportation Systems for the 2008 Olympics, IEEE Intelligent Systems, vol. 18, no. 6, pp. 8C11, 2003.
[17] Young-uk Chung and Dong-Ho Cho, Enhanced Soft-Handoff Scheme for Real-Time Streaming Services in Intelligent Transportation Systems Based on CDMA, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 7, NO. 2, pp.147-155, JUNE 2006.
[18] Joel C. McCall and Mohan M. Trivedi, Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 7, NO. 1, pp.20-37 MARCH 2006.
[19] Qing Wu and Ying Li. ScudADL: An Architecture Description Language for Adaptive Middleware in Ubiquitous Computing Environments. In proceedings of the 2th ISECS International Colloquium on Computing, Communication, Control, and Management, 2009.
[20] F.Oquendo, B. Warboys, R.Morrison, et al. ARCHWARE: Architecting Evolvalble Software, EWSA 2004, LNCS 3047, pp. 257-271, 2004.
[21] F.Oquendo, π-ADL: an architecure description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architecture, ACM SIGSOFT Software Engineering Notes, VOL. 29, NO. 3, pp.1-14, 2004.
[22] Li Changyun, Li Gansheng, he Pinjie, Formal Dynamic Architecture Description Language D-ADL, Journal of Sotware, VOL.17, NO.6, pp. 1349-1359, 2006.
[23] Jing Gao,Yuqing Lan,Shuhang Guo,Maozhong Jin,Tong Zhao.A new modeling method of component interaction behavior.Proceedings of the World Congress on Intelligent Control and Automation(WCICA), pp.4773-4778, 2008.

Wu Qing received the BS and MS degrees both in Computer Science from Hangzhou Dianzi University in July 2000 and March 2003, respectively. In June 2006, he received the Ph.D. degree in computer science from Zhejiang University. Since July 2007, he serves as an associate professor of computer science at Hangzhou Dianzi University. His major interests include Pervasive Embedded Computing, Software Middleware, Context-aware Computing, CORBA Component Model, CAA, and Multi-Agent Theory.

Chunbo Zhao is a graduate student of computer science at Hangzhou Dianzi University. His major interests include Distributed Computing, Software Middleware, and CORBA Component Model, and CAA Theory.

Yi Ying is an associate professor of computer science at Zhejiang University. His major interests include Pervasive Embedded Computing, Software Architecture, and Semantic Component Theory.