Component Dynamic Behavioral Compatibility Analysis of ScudWare Middleware for Ubiquitous Computing

Qing Wu, Chunbo Zhao
Institute of Computer Application Technology
College of Computer Science, Hangzhou Dianzi University
Hangzhou, Zhejiang, China
Email: wuqing@hdu.edu.cn

Abstract— **In ubiquitous computing environments, the software component dynamic behavior and its compatibility analysis are two important issues in middleware dynamic adaptation. In this paper, we firstly present an adaptive middleware architecture called ScudWare for a smart vehicle space. Then a semantic component model is given in detail. Next, for ScudWare middleware, we propose a semantic component dynamic behavior formalization and component behavior compatibility verification based on the higher-order π calculus. Next, a case study is given to evaluate our model and methods. Finally, we draw a conclusion and give our next work.**

Index Terms— **Ubiquitous Computing, Adaptive Middleware, Component Dynamic Behavior**

I. INTRODUCTION

A novel computing model called ubiquitous computing [1] is coming into our daily life. In ubiquitous computing environments, information and communication technology are anywhere, for anyone, and at anytime. The physical world and information space will gradually be united naturally and seamlessly. The smart space is considered as an integral implementation of ubiquitous computing, where the computing environment should continually adjust itself to deal with the situation changing. By using smart devices, users in this active computing environment can interact with the physical space transparently and seamlessly.

To realize the idea of ubiquitous computing, a lot of information and communication technologies should be developed and be integrated into our environments: from toys, desktops to rooms, factories and whole city areas with integrated processors, sensors, and actuators connected via wireless high-speed networks and combined with new output devices ranging from projections directly into the eye to large panorama displays.

In this new computing environments, users will naturally and transparently interact with each other and with entities in the space, and the space environment can automatically and continuously self-adjust to provide the better services for users. This attractive goal poses a large number of new challenges for software architecture and middleware technology. The traditional software infrastructure is no longer suitable for smart space [2]. As a result, a novel software platform that supports component dynamic adaptation is required.

Nowadays, the component technology has become one of the key technologies in the software industry, and it is a world trend to develop software products based on this technology. It's very active to research on the model of dynamic interaction of components, as well as the dynamic self-adaptive assembly and evolution of components in the field of component-based software development.

The remainder of the paper is organized as follows. Section 2 describes the ScudWare middleware platform including smart vehicle space, and the ScudWare middleware architecture. Then component dynamic behavior based on higher-order π calculus is proposed in section 3. Section 4 proposes the component behavior compatibility verification. In section 5, we give a case study of our methods. Next, some related work is stated in section 6. Finally, we draw a conclusion in section 7.

II. SCUDWARE MIDDLEWARE PLATFORM

Conformed to the CCM (CORBA Component Model) specification, we have built the ScudWare middleware platform [3] for smart vehicle space naturally and adaptively. We use the ACE (Adaptive Communication Environment) and the TAO (The ACE ORB). TAO is a real-time ORB (Object Request Broker) developed by Washington University. According to the application domain of smart vehicle space, we reduce the TAO selectively and add some adaptive services such as adaptive resource management service, context service, and notification service. ScudCCM, a part of ScudWare, is responsible for adaptive component management comprising component package, assembly, allocation, addition, removal, replacement, updating, and transfer. As following, we introduce

This paper is based on "ScudADL: An Architecture Description Language for Adaptive Middleware in Ubiquitous Computing Environments," by Qing Wu, and Ying Li, which appeared in the Proceedings of the 2009 IITA International Conference on Communication Systems, Networks and Applications (ICCSNA 2009), Sanya, China, August 2009. © 2009 IEEE.

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

doi:10.4304/jsw.5.12.1417-1424

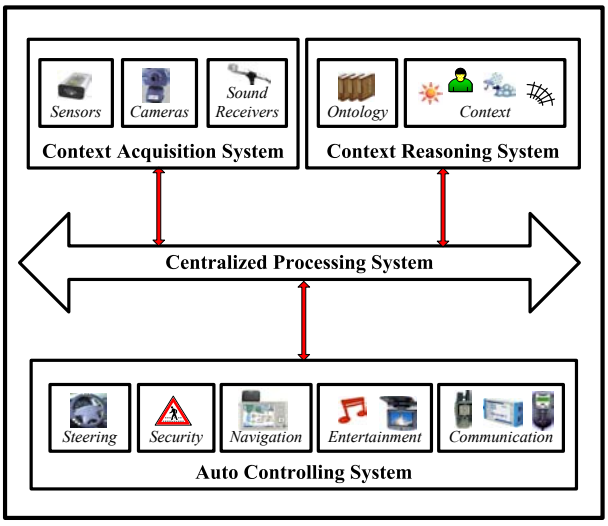

Figure 1. Smart Vehicle Space

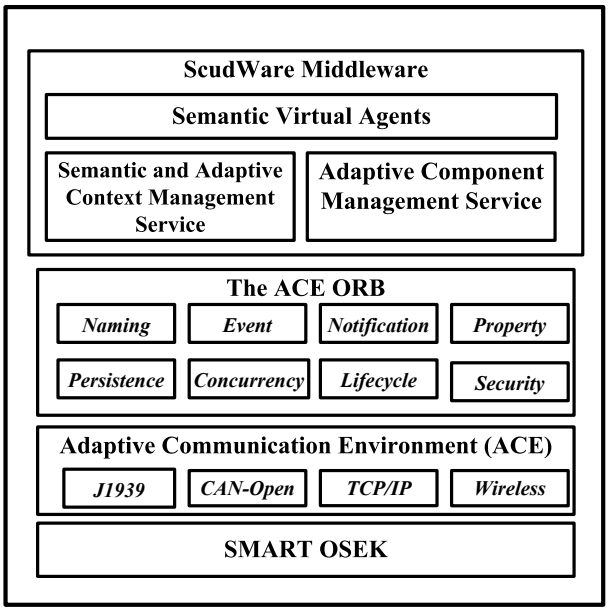

Figure 2. ScudWare Middleware Architecture

smart vehicle space, CCM specification and ScudWare architecture briefly.

A. Smart Vehicle Space

In recent years, a lot of developers have applied embedded, AI, and biology authentication technologies to vehicles. The drive capability, dependability, comfort, and convenience of the vehicle are improved greatly. When people go into smart vehicle space, they find many intelligent devices and equipments around them. They communicate with these tools naturally and friendly. It forms a harmonious vehicle space where people, devices, and environments co-operate with each other adaptively.

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.

B. ScudWare Middleware Architecture

As figure 2 shows, ScudWare architecture consists of five parts defined as *SCUDW* = (*SOSEK, ACE, ETAO, SCUDCCM, SVA*). *SOSEK* denotes SMART OSEK, an operating system of vehicle conformed to OSEK specification developed by us. *ACE* denotes the adaptive communication environment, 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 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 based on component dynamic behavior formalizations. In addition, the top layer is *SVA* that denotes semantic virtual agent [4]. *SVA* aims at dealing with application tasks. Each *sva* presents one service composition comprising a number of meta objects. During the co-operations of *SVA*, the SIP(Semantic Interface Protocol) [4] set is used including *sva* discovery, join, lease, and self-updating protocols. Due to the limited space, we don't detail *SVA* in this paper.

III. Component Dynamic Behaviors

In this section, we give a formalization of component dynamic behaviors. First, the overview of higher-order typed π calculus is given. Then we introduce the component dynamic behaviors modeling based on π calculus.

A. Overview of higher-order typed π calculus

The π calculus is one of formal methods to model and reason about concurrency and mobility. It extends CCS [5] with the ability to create and remove communication links between processes. One extension of the π calculus is higher-order typed π calculus by D.Sangiorgi [6], where the objects transmitted can also be processes. Process, name and abstraction are three parts of the higher-order typed π calculus. Process is a working unit of current running entity, and uses name to define channels and objects transmitted on the channel. Each process interacts with other process via a shared channel. We use P, Q, R, ... to range over processes. Name is a reference of one object. We use a, b, X, Y, ... to range over value names and object(process) names. Abstraction is a non-concrete process with some parameters. Thus, the class of processes is given by the following grammar.

$P ::= 0|\alpha(X).P|\overline{\alpha}(Y).P|P + Q|P|Q|(vX)P|[X = Y]P|A(\widetilde{K})$

(1) 0 is an empty process, which cannot perform any actions.

(2) $\alpha(X).P$ is an input prefix process. It means one name Z is received along one channel α, and X is a placeholder for the receive name. After this input, it will continue as process P and X will be replaced by the newly received name Z, which is described as $P[Z/X]$.

(3) $\overline{\alpha}(Y).P$ is a output prefix process. It means the name Y is sent along the channel α, and thereafter the process continues as P.

(4) $P+Q$ is a sum process, which represents a process that can either P or Q.

(5) $P|Q$ is a parallel composition process, which represents the combined behavior of P and Q executing in parallel. P and Q can act independently, and may also communicate if one performs an output and the other an input along the shared channel.

(6) $(vX)P$ is a restriction process. The process behaves as P, but cannot use the name X to communicate with other processes since X is a local name in P.

(7) $[X = Y]P$ is a match process. If X and Y are the same name, the process will behave as P, otherwise it dose nothing.

(8) $A(\widetilde{K})$ is an abstraction with concrete parameters process. A is an abstraction, defined as $\widetilde{X}A$. $\widetilde{X}$ is a set of process formal parameters, and $\widetilde{K}$ is a set of process actual parameters. So $A[(\widetilde{K})/(\widetilde{X})]$ is conducted.

In addition, process reduction and transition rules are defined in π calculus with particular semantics. The reduction rules consists of $R-COM$, $R-PAR$, $R-RES$, and $R-STRUCT$.

B. ScudADL Framework

ScudADL [9] extends D-ADL [7] and π-ADL [8], which can describe structure and behavior characters of adaptive middleware. Different with other ADLs, in ScudADL, component inner and outer adaptive behaviors are separated from component functional behavior in an explicit way. In addition, ScudADL can provide component resources interfaces describing computing resources requirement, consume, and available information, which is separated from component port. The component resources interfaces are attached via resource connecters. As following, we introduce its functional modules in turn.

In ScudWare architecture, components are essential software entities. The common 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 common 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 common components. As a result, the common and system components are executors of functional and non-functional behaviors of ScudWare middleware.

Ports are connection points of components communication. When component A want to send value message to component B, one port of A and one port of B will be used to build a communication link called a channel. Connector is a special component and responsible for channel management. Therefore, channels are dynamic built by the connector and conduct components routing actions. In addition, component A can send adaptive logic process to component B based on the process passing in higher-order typed π calculus.

Resource interface can communicate with computing environments component, show and operate its component computing resource. After connecting resource interfaces, it build a resource channel administered by resource connector, which is a special channel to deal with component resource requirement and consumption. In terms of different component resource consumption, the component will provide different execution quality for other components. At one period of time, the computing resources for one component are variable. So the interactions of components will show different effects such as satisfying or unsatisfying effects. Once the computing resources cannot satisfy the component minimal resource requirement, the interactions between this component and other component will be halted. It brings a bad executing effect for the middleware system. Via resource channel, one component transmit a execution model with one special quality required from another component.

We use higher-order typed π calculus as a foundation, extending D-ADL and π-ADL, to build a ScudADL describing dynamic behavior semantics for adaptive middleware system in ubiquitous computing environments. In ScudADL, there are two kinds of types those are base type and constructed type. The base type consists of any, natural, integer, real, boolean, string, and action type. The action type consists of condition, choose, compose, decompose, replicate, and send or receive object action. The constructed types consists of channel type, resource

channel type, behavior type, component type, connector type, resource connector type. Here, behavior type is a sequence of the action type, including component functional behavior, connector routing behavior, and inner or outer adaptive behavior type. Behavior type corresponds to process of higher-order typed π calculus. The component type and connector type correspond to abstraction of higher-order typed π calculus.

C. Semantic Component Formalization

Here, we use ScudADL to describe the semantic component model.

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.

7) $\widetilde{IAdapBeha} ::= < \widetilde{ChangeExeModel} > .\widetilde{IAdapBeha}$. 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) $OAdapBeha ::= AddAc.OAdapBeha | RemoveAc.OAdapBeha | UpdateAc.\ OAd\ apBeha | ReplaceAc.OAdapBeha | inaction$. 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.

D. Component Dynamic Behaviors Modeling

The 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 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 S, R, and $SRandRS$ 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 example, one network-based commodities trading system in ubiquitous computing environments have two essential components those are seller component and purchaser component. We firstly describe the seller component, which has three input channels: 1) contact channel for contacting with purchaser, 2) order channel

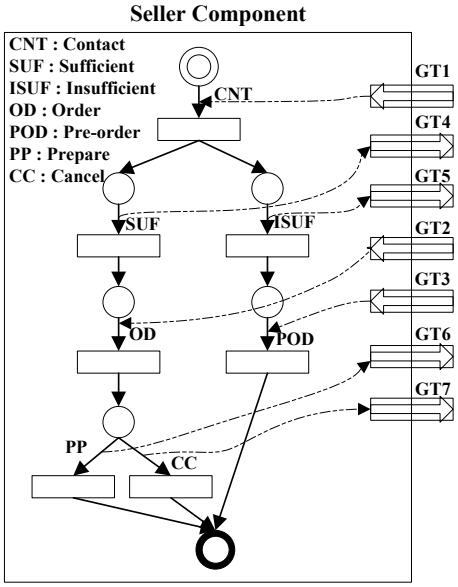

Figure 3. The seller component behavior view

for receiving order, and 3) preorder channel for receiving preorder. In addition, this component has four output channels: 1) sufficient channel for using when inventory meets demands, 2) insufficient channel for using when inventory can't meet demands, 3) prepare channel for allocating goods, and 4) cancel channel for canceling allocating goods. The seller component dynamic behaviors are illustrated in figure 3.

Then we give the definitions of every input and output channel. Input channels including $Contact(CNT)$, $Order(OD)$, $Preorder(POD)$ are one by one defined as $GT1$, $GT2$, and $GT3$. In addition, output channels including $Sufficient(SUF)$, $Insufficient(ISUF)$, $Prepare(PP)$, and $Cancel(CC)$ are one by one defined as $GT4$, $GT5$, $GT6$, and $GT7$.

According to the basic grammar of π calculus, the behavior of inputting CNT can be described as $GT1?(CNT)$. Similarly, the behavior of inputting OD, $POD4$, SUF, $ISUF$, PP, and CC can be in turn described as $GT2?(OD)$, $GT3?(P\ OD)$, $GT4?(SUF)$, $GT5?(ISUF)$, $GT6?(PP)$ and $GT7?(CC)$.

We give the definition of this seller component behaviors as PSC. When the inventory meets demands, it will execute process $PSUF$, otherwise it will execute process $PISUF$. In terms of above definitions, the seller component behaviors can be described as follows.

$P_{SC} = GT1?(CNT) \cdot (P_{SUF} + P_{ISUF})$

$P_{SUF} = GT4!(SUF) \cdot GT2?(OD) \cdot (P_{PREPARE} + P_{CANCEL})$

$P_{PREPARE} = GT6!(PP) \cdot 0$

$P_{CANCEL} = GT7!(CC) \cdot 0$

$P_{ISUF} = GT5!(ISUF) \cdot GT3?(POD) \cdot 0$

IV. Component Behavior Compatibility Analysis

In this section, we will give some definitions of component behavior compatibility and introduce its verification.

A. Component Dynamic Behavior Compatibility

In a component-based ubiquitous computing system, the component behavior compatibility plays an important role. This compatibility will reflect whether the components' interactive behaviors can be normally completed or affect the executing stability and feasibility of the whole system. In the following, we give some definitions of component behavior compatibility.

Definition 1: Feasible interactive path(FIP). Assume that component CA and component CB have interactive behaviors: CA sends message msg_1 to CB through channel gt_{a1}, then CB receives msg_1 from CA through channel gt_{b1} and sends feedback msg_2 to CA through channel gt_{b2}. At last, CA receives msg_2 from CB through channel gt_{a2}. If this interactive behavior can be successfully completed by inputting and outputting messages, then the sequence of msg_1 and msg_2 is called a feasible interactive path. Otherwise, this sequence of msg_1 and msg_2 is called an infeasible interactive path.

$FIP \overset{msgsequence}{\longrightarrow} Finish(interactive \cdot behavior) = Succ$

Definition 2: Absolutely compatible(AbComp). In component CA and component CB, if each interactive path(IPath) is a feasible interactive path, then CA and CB are absolutely compatible.

$AbComp(CA, CB) \longrightarrow \forall IPath \in (CA \bigcap CB) \cdot IPath = FIP$

Definition 3: Relatively compatible(ReComp). Component CA and component CB is not absolutely compatible. And they have at least one feasible interactive path. Then CA and CB are relatively compatible.

$ReComp(CA, CB) \longrightarrow (AbComp(CA, CB) = false) \wedge (\exists IPath \in (CA \bigcap CB) \cdot IPath = FIP)$

Definition 4: Compatibility degree(CoDeg). As for interactive behaviors of component CA and CB, if the count of feasible interactive paths is m, and the count of all interactive paths is n, then the component compatibility degree of CA and CB is m/n.

We can evaluate the compatibility of component interactive behaviors according to above definitions. In general, the compatibility of components can be absolutely compatible or relatively compatible. However systems in ubiquitous computing environments, those having high requirement of stability, correctness and the behavioral compatibility of components, always need high compatibility of different components or exactly need to be absolutely compatible.

B. Component Compatibility Analysis

It always relates to behavioral interaction of multiple components when verifying their behavioral compatibility. Now we explain the general regulars by considering two components. The general regular for compatibility verification of more than two components can be made by giving an analogy.

According to the different transfer forms of messages, the atomic interactive behaviors of two components can be divided into three kinds: 1) all send only, 2) all receive

only, and 3) one send one receive. The behaviors of two components are described as P_1 and P_2, and the transferred messages are described as m_1 and m_2. The interactive behaviors can be described by concurrent processes. The atomic interactive behaviors of components can be described as follows:

(1) $P_1||P_2 = gt_1!(m_1)\cdot 0 \| gt_2!(m_2)\cdot 0$. Two components send messages only but these messages will never be received and dealt with. At last the interaction can't complete normally, so these components aren't compatible.

(2) $P_1||P_2 = gt_1?(m_1)\cdot 0 \| gt_2?(m_2)\cdot 0$. Two components receive messages only but no messages can be obtained. Because no feasible interactive path can be found, they aren't compatible either;

(3) $P_1||P_2 = gt_1!(m_1)\cdot 0 \| gt_2?(m_2)\cdot 0$.

(4) $P_1||P_2 = gt_1?(m_1)\cdot 0 \| gt_2!(m_2)\cdot 0$.

As for (3) and (4), one component send message, while another receive. If the message sent by P_1 is the same as the message received by P_2, the interactive behavior of these two components can make a further evolution:

$P_1 \| P_2 \xrightarrow{m_1 \vee m_2} 0 \| 0 = 0$

It is obvious that each component will complete its action after the message is transferred, thus they are behavior compatible. The feasible interactive path is a message sequence of m_1 and m_2. However, if m_1 is not equal to m_2, each component will not complete its action successfully and at last they will not be compatible.

Specially, the above three kinds of atomic interactive behaviors can be extended to various complex interaction of components.

V. Case Study

In a network-based commodities trading system in ubiquitous computing environments, according to the seller component mentioned above, we can create a purchaser component. Then we combine them into an entirety and verify the compatibility of their interactive behaviors.

The purchaser component has seven channels, including three output channels and four input channels. The three output channels are Contact (a channel for contacting with seller), Order (a channel for sending order) and Preorder (a channel for sending preorder). The four input channels are Sufficient (inventory meet demand), Insufficient (inventory can't meet demand), Prepare (allocate goods) and Cancel (cancel allocate goods). Similarly, every input and output channel are named. The behavior view of purchaser component is concretely described in figure 4.

If P_{PC1} represents the behavior of purchaser component, then it can be described as follows:

$P_{PC1} = GT1!(CNT)\cdot(P_{SUF1} + P_{ISUF1})$

$P_{SUF1} = GT4?(SUF)\cdot GT2!(OD)\cdot(P_{PREPARE1} + P_{CANCEL1})$

$P_{PREPARE1} = GT6?(PP)\cdot 0$

$P_{CANCEL1} = GT7?(CC)\cdot 0$

$P_{ISUF1} = GT5?(ISUF)\cdot GT3!(POD)\cdot 0$

After that, the seller component and purchaser component can be integrated into an entirety, as is shown in figure 5.

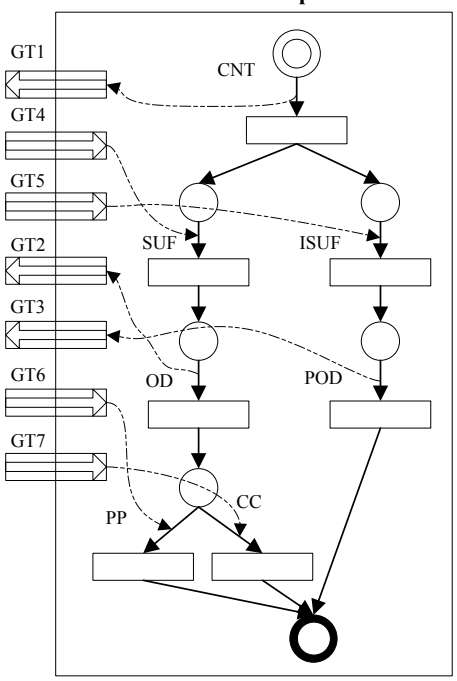

Figure 4. The purchaser component behavior view

The dynamic interaction between components can be seen as a concurrent system, and it also can be described by the concurrent processes. The concurrent interactive behaviors of seller component and purchaser component can be described by π calculus as follows.

$P_{(SC,PC1)} = P_{SC} \| P_{PC1} =$

$P_{SC} \| P_{PC1} = (GT1?(CNT)\cdot(P_{SUF} + P_{ISUF}) \| (GT1!(CNT)\cdot(P_{SUF1} + P_{ISUF1}))$

The component evolution is shown as follows.

$P_{(SC,PC1)} = P_{SC} \| P_{PC1} =$

$(GT1?(CNT)\cdot(P_{SUF} + P_{ISUF}) \| (GT1!(CNT)\cdot(P_{SUF1} + P_{ISUF1}))$

$\xrightarrow{CNT} (GT4!(SUF)\cdot GT2(OD)\cdot(P_{PREPARE} + P_{CANCEL}) + P_{ISUF}) \| (GT4?(SUF)\cdot GT2!(OD).(P_{PREPARE1} + P_{CANCEL1}) + P_{ISUF1})$

$\xrightarrow{SUF} (GT2?(OD)\cdot(P_{PREPARE} + P_{CANCEL})) \| (GT2!(OD)\cdot(P_{PREPARE1} + P_{CANCEL1}))$

$\xrightarrow{OD} (GT6!(PP)\cdot 0 + GT7!(CC)\cdot 0)$

As we can see, the concurrent processes become a null process at last. Thus the sequence of CNT, SUF, OD and PP is a feasible interactive path. Likewise, the sequence of CNT, SUF, OD, and CC, and the sequence of CNT, $ISUF$, POD are also feasible interactive paths. Besides the above three interactive paths, there is no other interactive path. So all the interactive paths are feasible interactive paths and the interactive behavior of the two components is absolutely compatible.

In figure 5, the purchaser component will send POD to the seller component when it has received $ISUF$ from the seller component. If we remove this action, and the purchaser component will not receive PP from the seller component. Then the interactive behavior view of seller

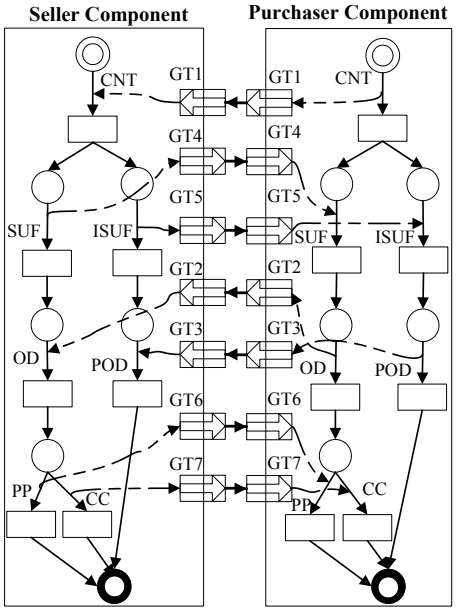

Figure 5. The interactive behavior view of seller and purchaser components

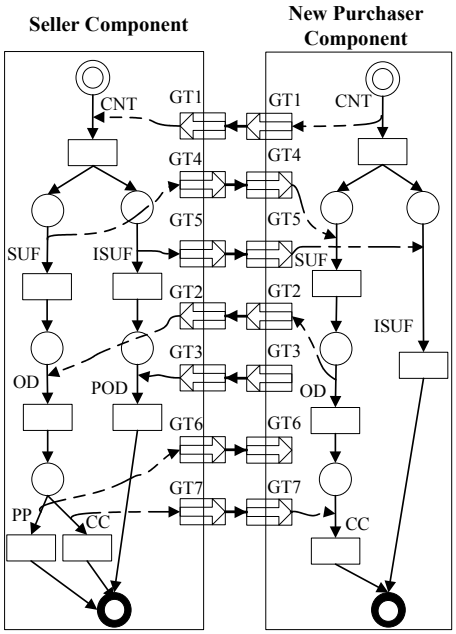

Figure 6. The interactive behavior view of seller component and new purchaser component

component and new purchaser component is concretely described in figure 6.

If P_{PC2} represents the behavior of new purchaser component, then it can be described as follows.

$P_{PC2} = GT1!(CNT) \cdot (P_{SUF2} + P_{ISUF2})$

$P_{SUF2} = GT4?(SUF) \cdot GT2!(OD) \cdot P_{CANCEL2}$

$P_{CANCEL2} = GT7?(CC) \cdot 0$

$P_{ISUF2} = GT5?(ISUF) \cdot 0$

The concurrent interactive behavior of seller component and new purchaser component can be described by π calculus as follows.

$P_{(SC,PC2)} = P_{SC} \| P_{PC2}$

$P_{SC} \| P_{PC2} = (GT1?(CNT) \cdot (P_{SUF} + P_{ISUF}) \| (GT1!(CNT) \cdot (P_{SUF2} + P_{ISUF2}))$

The component evolution is shown as follows.

$P_{(SC,PC2)} = P_{SC} \| P_{PC2}$

$= (GT1?(CNT) \cdot (P_{SUF} + P_{ISUF}) \| (GT1!(CNT) \cdot (P_{SUF2} + P_{ISUF2}))$

$\xrightarrow{CNT} (P_{SUF} + GT5!(ISUF) \cdot GT3?(POD) \cdot 0) \| (P_{SUF2} + GT5?(ISUF) \cdot 0))$

$\xrightarrow{ISUF} (GT3?(POD) \cdot 0 \| 0)$

In terms of above description, the component evolution shows that after the seller component sends $ISUF$ to the new purchaser component, it will never receive POD from the new purchaser component, and the interaction will not successfully complete at last. However, a feasible interactive path of CNT, SUF, OD, CC exists. Thus the interactive behavior of two components is relatively compatible. After that, let us calculate the compatibility degree of the seller component and the new purchaser component. There are three interactive paths between them, which are 1) CNT, SUF, OD, PP, 2) CNT, SUF, OD, CC, 3) CNT, $ISUF$, POD. Only one feasible interactive path exists, which is CNT, SUF, OD, CC. As a result, the compatibility degree is 1/3.

VI. Related Work

In recent years, many efforts have be made to design the new middleware architecture capable of supporting smart spaces in ubiquitous computing.

The Stanford Interactive Workspaces project [10] aims at exploring new possibilities for people to work together in technology-rich spaces with computing and interaction devices on many different scales. This project concentrates on task-oriented work such as brainstorming meetings and design reviews. They have developed iROS [11] , a middleware platform for a class of ubicomp environments, through the use of three guiding principles - economy of mechanism, client simplicity and levels of indirection. RCSM [12](Reconfigurable Context- Sensitive Middleware) is designed to facilitate applications that require context awareness or spontaneous and ad hoc communication.

In addition, we have obtained some achievements on how to formalized express component behavior, how to verify the consistency and compatibility of component interactive behaviors. Gao [13], Xue [14], and Zhong [15] have used Pi-calculus to model behavior, but they haven't verified the equivalence of different behaviors. Shen [16], Lucia [17], and Hu [18] have made some research on the consistency of components' dynamic evolution, but it can't ensure the correctness of internal flow structure and the in existence of deadlock or inaccessible state. In order to ensure the stability and normality of the whole system after creating, deleting, replacing or recombining components, and improve the self-adaptability of the dynamic evolution of components, we must ensure the interactive compatibility of components. Those incompatible components always lead to the collapse of the whole system. Aiming at this problem, this paper emphatically

verifies the compatibility of interactive behaviors among components.

VII. Conclusions and Future Work

In this paper, we present an architecture of ScudWare middleware. Then for this middleware, we propose the component dynamic behavior formalization and component behavior compatibility verification based on the higher-order π calculus. Next, a case study is given to evaluate our methods.

Through the compatibility verification of components' interactive behavior, we can effectively ensure the stability and normality of the whole system's dynamic adaptation after increasing, deleting, replacing, transferring, and recombining of component. Our next work include that 1) we will make further research on the behavioral compatibility of component. 2) regarding as component compatibility degree, the running efficiency and stability of system should be synthetically considered.

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] Anand Tripathi, Next-Generation Middleware Systems Challenges Designing. *Communications of the ACM*, 45(6), pp. 39-42, 2002.

[3] 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.

[4] 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.

[5] R.Milner, J.Parrow, D.Walker, A Calculus of Mobile Processes. Infromation and Computation, VOL.100, No.1, pp.1-40, 1992.

[6] D.Sangiorgi, Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD Thesis, University of Edinburgh, 1992.

[7] Li Changyun, Li Gansheng, he Pinjie, Formal Dynamic Architecture Description Language D-ADL, Journal of Sotware, VOL.17, NO.6, pp. 1349-1359, 2006.

[8] 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.

[9] 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.

[10] Stanford Interactive Workspaces Project. http://iwork. stanford.edu/

[11] Shankar R. Ponnekanti, Brad Johanson, Emre Kiciman and Armando Fox, Portability, Extensibility and Robustness in iROS, *Proc. IEEE International Conference on Pervasive Computing and Communications*, March 2003.

[12] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K.S. Gupta, Reconfigurable Context-Sensitive Middleware for Pervasive Computing, *IEEE Pervasive Computing*, pp.33-40, July-September 2002.

[13] Gao Jing, Lan Yuqing, Guo Shuhang, Jin Maozhong, Zhao Tong. A new modeling method of component interaction behavior. Proceedings of the World Congress on Intelligent Control and Automation (WCICA), pp.4773-4778, 2008.

[14] Xue Gang, Yao Shaowen, Joan Lu. Workflow model description based on Pi-calculus. Computer science, 35(7):191-194, 2008.

[15] Zhong Hui, Wang Weiping, Huang Yantan, Li Qun. A Pi-calculus based behavior modeling method. Theory and practice of systems engineering, 29(5): 175-185, 2009.

[16] Shen Limin, Ma Chuan, Wang Tao. A research on consistency of dynamic evolution of components based on Pi-calculus. Application research of computers, 26(4): 1345-1348, 2009.

[17] Lucia Acciai, Michele Boreale. Spatial and Behavioral Types in the Pi-Calculus.Lecture Notes in Computer Science 5201, pp.372-386, 2008.

[18] Hu HY, Lv J. Study on behavioral compatibility of components in software architecture using object-oriented paradigm. Journal of software, 17(6):1276-1286, 2006.

Qing Wu 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.