A Semantic and Adaptive Middleware Architecture for Pervasive Computing Systems

Wu Qing, Hu Weihua
College of Computer Science, Hangzhou Dianzi University
Email: {wuqing, hwh}@hdu.edu.cn
Ding Wen
Zhejiang Education Examine
Email: dwzz11@mail.hz.zj.cn

Abstract— **With the increasing demands for adaptive middleware of dynamic systems in pervasive computing environments, the need for dynamic software architecture and programming infrastructure to achieve dynamic adaptation is widely recognized. In this paper, we firstly present a semantic and adaptive middleware architecture called ScudWare that supports for dynamic and heterogeneous environments. ScudWare middleware is based on adaptive communication environments, which consists of adaptive components, and semantic virtual agents. Specially, a ScudADL framework and the specification semantics, based on higher-order typed π calculus theory, are proposed, which describes ScudWare component structure characters, and dynamic behavior adaptation. In the ScudADL, the computing resources consumption is concerned. And the component inner and outer adaptive behaviors are separated from component functional behaviors in an explicit way. Finally, we introduce an application of ScudWare architecture, which is a computer aided assessment system in a smart CAA space, and give a case study to show its adaptation.**

Index Terms— **adaptive middleware, smart CAA space, architecture description language, higher-order typed π calculus**

I. INTRODUCTION

Today pervasive computing [1] as a novel computing model is coming into our daily life. In pervasive 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 pervasive computing, where the computing environments 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 above computing model, a lot of information and communication technologies should be developed and be integrated into our environments: from toys, desktops to rooms, factories and the 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.

Nowadays many efforts are made for pervasive computing [2]–[4], especially in smart spaces, e.g., the smart home [5], [6], the Aware room [7], and the smart museum [8]. Smart room acts like an invisible butler. It has cameras, microphones, and other sensors, and uses these inputs to understand what people are doing in order to help them. It can recognize who is in the room and can interpret his or her hand gesture, and interior that know when drivers are trying to turn, stop, pass, etc., without being told. The aware home aims at creating a living laboratory for research in pervasive computing for everyday activities, which has two identical and independent living spaces, consisting of two bedrooms, two bathrooms, one office, kitchen, dining room, living room and laundry room. In addition, there will be a shared basement with a home entertainment area and control room for centralized computing services. Smart museum is a fertile ground for studying systems to enhance the visitor's experience, and tries to improve on this experience with multimedia content. In smart museum, static and local contents are used on a PDA, including a short video clip of the artist explaining his or her work.

In the smart spaces, 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 [9] for lack of adaptation mechanism. Therefore a novel software middleware architecture is required to meet the requirements of pervasive computing.

This paper addresses the middleware platform and its architecture description language (ScudADL) for dynamic adaption in smart spaces. We have developed a semantic and adaptive middleware platform called *ScudWare* [10], which is based on semantic information and conformed to a lightweight CCM (CORBA component model) spec-

This paper is based on "Computer Ability Assisted Assessment System for Large-Scale Heterogeneous Distributed Environments," by Wu Qing, Hu Weihua, Zhou Bishui, Ding Wen and Chen Tianzhou, which appeared in the Proceedings of the 9th International Conference for Young Computer Scientists(ICYCS 2008) Zhangjiajie, China, November 18-21, 2008. © 2008 IEEE.

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

doi:10.4304/jsw.4.10.1061-1068

ification [11].

The rest of the paper is organized as follows. Section 2 presents a ScudWare middleware platform including CCM specification overview, its architecture, and a smart CAA space. Section 3 proposes a Scud architecture description language, consisting of higher-order typed π calculus overview, the ScudADL framework, and ScudADL formulation semantics. In section 4, we introduce a computer grade examination of Zhejiang Province, which is an application of pervasive computing systems in a smart CAA space. In section 5, we give a case study of cartoon study program in the smart CAA space. Then some related work is stated in section 6. Finally, we draw a conclusion and give the next work in section 7.

II. ScudWare Middleware Platform

In this section, we give a CCM specification overview firstly. Then a ScudWare architecture is presented. At last, we introduce a smart CAA space in pervasive computing environments.

A. CCM Specification Overview

CORBA (Common Object Request Broker Architecture) is one of software middlewares, which provides language and operating system independences. CCM is an extension to CORBA distributed object model. CCM prescribes the specifications of component designing, programming, packaging, deploying and executing stages.

CCM specification defines component attributes and ports. Attributes are properties employed to configure component behavior. Specially stated, component ports are very important, which are connecting points between components. There are four kinds of ports: facets, receptacles, event sources, and event sinks. Facets are distinct named interfaces provided by component for client interaction. Receptacles are connection points that describe the component's ability to use a reference supplied by others. Event sources are connection points that emit events of a specified type to one or more interested event consumers, or to an event channel. Event sinks are connection points into which events of a specified type may be pushed.

In addition, CCM specification defines component home, which is a meta-type that acts as a manager for component instances of a specified component type. Component home interfaces provide operations to manage component lifecycle. CIF (Component Implementation Framework) is defined as a programming model for constructing component implementations. CIDL (Component Implementation Definition Language), a declarative language, describes component implementations of homes. The CIF uses CIDL descriptions to generate programming skeletons that automate many of the basic behaviors of components, including navigation, identity inquiries, activation, state management, and lifecycle management. The component container defines run-time environments for a component instance. Component implementations may be packaged and deployed. A CORBA component package maintains one or more implementations of a component. One component can be installed on a computer or grouped together with other components to form an assembly.

B. ScudWare Middleware Architecture

ScudWare architecture consists of five parts defined as *SCUDW* = (*ACE, ETAO, SCUDCCM, SVA*). *ACE* denotes the adaptive communication environment [12], 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 [13] 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 [14]. *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.

C. Smart CAA Space

CAA is a computer aided assessment, use computing devices to assess the users' computer operation ability and technique. It can increase the frequency of assessment, increase the range of knowledge assessed and increase feedback to students and staff. In addition, it can expand the range of assessment methods, increase objectivity and consistency, and decrease marking loads.

Smart CAA space shown as figure 1 includes users, computing devices, management service of computer knowledge and operation technology , and study and assessment platform. The users consist of students and teachers. The computing devices are fixed or mobile, such as desktop computer,laptop computer, and PDA etc. CAA management service is responsible for providing computer knowledge and operation technology in a proper way on the computing device for users' requirements. The study and assessment software platform is a adaptive software infrastructure that provides a good software environment for study and assessment.

III. SCUD Architecture Description Language

In this section, we present a SCUD architecture description language. Firstly, a overview of high-order typed

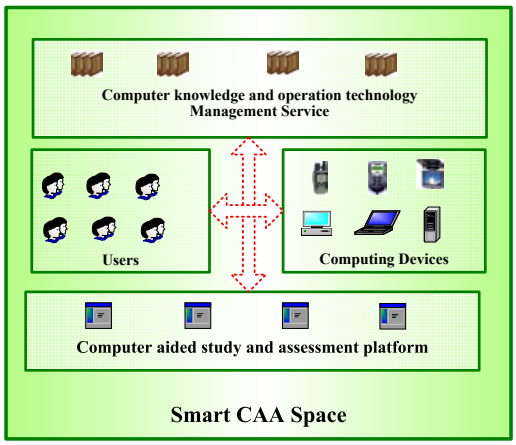

Figure 1. Smart CAA Space

π calculus is given. Secondly, we propose a ScudADL framework. Finally, the ScudADL formalization and semantics are detailed.

A. Overview of high-order typed π calculus

The π calculus is one of formal methods to model and reason about concurrency and mobility. It extends CCS [15] 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 [16], 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.

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

(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) $(\upsilon X)P$ 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 extends D-ADL [17] and π-ADL [18], 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. The structure of ScudADL framework is shown in figure 2. 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

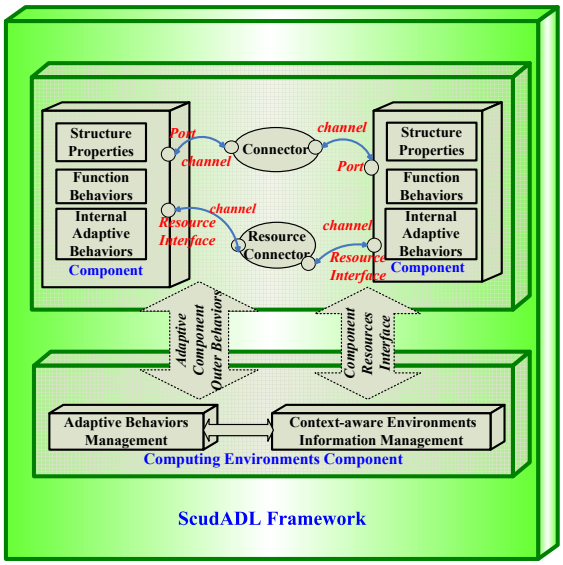

Figure 2. ScudADL Framework

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 pervasive 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. ScudADL Formalization

In this section, we use ScudADL to define adaptive component, describing its structure character and dynamic behavior.

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

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

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

3) $< \widetilde{RI} >$ 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.

4) $< \widetilde{ExeModel} >::== (< \widetilde{Res}, \widetilde{ExeQua} >)$. 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.

5) $< \widetilde{FuncBeha} >::==< \widetilde{IO} >< \widetilde{FuncBeha} > | < \widetilde{Condition} >< \widetilde{FuncBeha} > | < \widetilde{Choose} >< \widetilde{FuncBeha} > |unobservable|inaction$. 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

6) $IAdapBeha$ ::=$< \widetilde{ChangeExeModel} >$ $.IAdapBeha$. 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 $((RC^p_{cc}, RC^q_{cm}, RC^r_{mm}), EQ^{\mu}_{ac})$.

7) $OAdapBeha$ ::= $AddAc.OAdapBeha$ | $RemoveAc.OAdapBeha$ | $UpdateAc.OAdapBeha$ | $ReplaceAc.OAdapBeha$ | $inaction$. 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 a 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 ($P \approx Q$).

IV. Computer Aided Assessment System

This section will give a system overview of the computer aided assessment system in smart CAA space. Then we present some implementation technologies of the system.

A. System Overview

With the continuous deepening of teaching reform, the development of modern long-distance teaching , and the network assisted teaching and examination, which is an important mean to test the student's ability and the teaching quality, also should be reformed. At present an effective measure is the development of distributed computer ability assessment system. "Ministry of Finance on the implementation of schools of higher learning and teaching quality of undergraduate education reform projects" (the high education 2007 No.1 file) accentuate that we should develop the online exam systems, formulate the relevant standards, progressively realize the on-line examination of college English and the examination courses for National Network Education, and create a safe, convenient, and efficient platform for the examination.

Today the network-based examination as a new form of modern education reform gradually shows its potential advantages. Because the system is in a smart CAA space that is a large-scale heterogeneous distributed environment, it should be continuously improved its functionality, security and stability characteristics according to the changes.

We have developed a computer aided assessment system based on the ScudWare middleware, which has a comprehensive ability to analyze the different examination types, questions and environments. It can also abstract the semantic information expression for the examination, and then establish CAA(Computer Aided Assessment) [19] evaluation integration language. For the characteristics of the wide region and the large-scale in the distribution node of the examination, we present a separation of concerns method, and use the semantic virtual agents to monitor the examination nodes. The examination stream is used in the system, which is combined with the examination results and the problem-solving process, and shows the tracking of the examination process. In addition, we propose an intelligent examination fusion algorithm for the examinees' operation process, which can evaluate the examinees' operation ability. In order to evaluate the examinees' mental results in design examination, we use a reconstruction technology, which is based on pattern recognition and virtual environment information. We also present a comprehensive evaluation method according to the implementation correctness, the correctness of code and the correctness of output [20] [21], then establish a comprehensive evaluation system.

B. Implementation Technologies

In the system, many implementation technologies are used to support the large-scale heterogeneous distributed environments, include distributed concurrent processing, online real-time monitoring, and intelligent examination evaluation.

1) Distributed Concurrent Processing: During the computer grade examination, each school should arrange computer rooms, and organize students for examination. Each school is an examination unit, As a result, there

must be at least one high performance computers used as the servers, which can be centralized as the main servers for online testing for multiple terminals at the same time. The server must be able to deal with large number of large-scale data from terminal's request or response. Meanwhile the server must be able to process timely and real-time information. The system uses a distributed concurrent processing technology to handle the large-scale information. By this, the system has some characteristics. 1) The distributing range expends from the narrow space of the computer frame to a bigger space of the local area network. 2) The global operating system can control each networking computers. Each networking computer's user can see a distributed computer system that supports the distributed multi-user in the distributed environments.

2) Online Real-time Monitoring: The system can monitor the examinees' activities by socket real-time connection. Examiners and the examinees can two-way exchange information. Once the network interrupts abnormally, examinees can continue testing in the case that resumption of network can automatically restore your connection. In addition, the online real-time monitoring technology can conduct lock-on examinees, mandatory paper submitted, dissemination of information, the re-examination, and other managements.

Furthermore the server can monitor examinees' violation of regulations. For example, once the examinee copy the files on different computers, this information is monitored by the server. In addition, through monitoring examinees' screens, the examiners are all in control of examinees' every activity. As a result, examiners may at any time monitor the relevant actions of examinees by screen monitor.

To prevent duplication of examinations, exchange, inter-copy, and modified examination results have been fully taken into account in the system. By perfect examination room invigilators logic, the system can effectively eliminate fraud behavior in the examination room, and maintain examination fairness.

3) Intelligent Examination Evaluation: At present, the question types in the computer grade examination consist of blank-filling, program modify and program design questions. In order to achieve intelligent examination evaluation, we use two evaluation methods. 1) For Blank-filling questions, a statement similarity evaluation mechanism is proposed to achieve automatically correcting technology strategy. Statements similarity is used to evaluate students answer with the standard answer to the close of its main consideration in semantics between the two on the degree of similarity, but the grammatical structure does not make too much consideration to their specific ideas are as follows. a) Using a technology similar to compiled technology to morphology and syntax analysis for the answers that will be broken down into its individual identifier, constant and operator, and semantics by partially as a whole. b) After generation semantic equivalence class for standard answers, the programming language in a phrase or expression may have more than one form of equivalence. c)After morphology, grammar analysis and optimization for examinees' answers, the answer can be viewed as a correct answer if the answer matching degree is more than 95 percents. 2) For program modify and design question, most of the current processes those are smart markers by running candidates prepared by generating a corresponding output file, and then to compare documents and standards. According to results of this comparison, a corresponding scores is given. However, this method could not give a reasonable score for actual programming ability of the examinees. As a result, our system use residual frame algorithm, taking into account the implementation confidence, credibility and the output code credibility, to assess the examinees' programming abilities.

V. Case Study

In this section, we will give a case study of smart CAA space in computer aided system, and using ScudADL to describe the component structure and dynamic behavior, as shown in figure 3

In a cartoon design study program, one student use a PDA to communicate with the knowledge study server via wireless network. On the one hand, there is a cartoon video component called A_c^c in the student's PDA, which can get cartoon video information from the server, and show them in real time to achieve a satisfying effect. On the other hand, there is a cartoon source generator component called A_c^s in the knowledge study server, which can provide the cartoon video information for a identity valid user.

During the period of 0 to t_1, all computing resources for A_c^c and A_c^s are satisfied. A_c^c consume RC_{cc}^i, RC_{cm}^j, and RC_{mm}^k computing resources, and acquire a good quality cartoon video information from A_c^s. And the execution model is $ExeModel_{hq}$ that is with high quality. However, at time t_1, the wireless network bandwidth decrease, and the power of the PDA decrease to 40%. In order to continue this communication and finish the cartoon design study program, during the period of t_1 to t_2, the A_c^s will be replaced by A_c^{s1} that sends low quality cartoon video information for the client after an $OAdaptBeha(ReplaceAc)$ behavior is conducted. In addition, the A_c^c will adjust to consume RC_{cc}^p, RC_{cm}^q, and RC_{mm}^r computing resources, and change its execution model that shows the cartoon video on a dark screen for lower power consumption after an $IAdapBeha(ChangeExeModel)$ behavior is conducted. And at t_2, the execution model is $ExeModel_{lq}$ that is with low quality and can continue the application. By these methods, the system can continue running well to need the dynamic computing environments.

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.

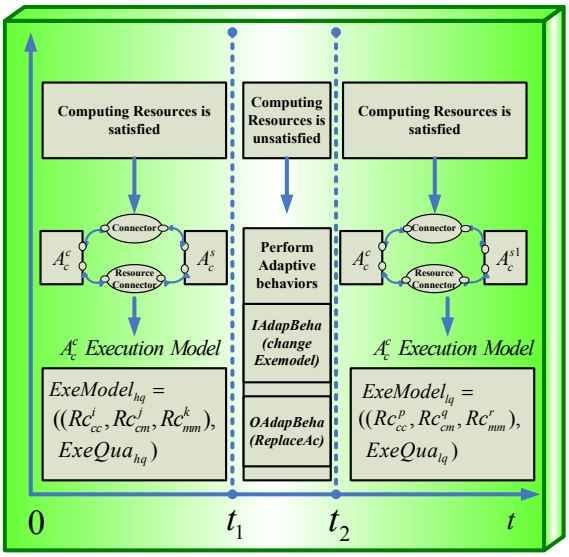

Figure 3. Case Study

The Stanford Interactive Workspaces project [22] 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 [23] , 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 [24](Reconfigurable Context- Sensitive Middleware) is designed to facilitate applications that require context awareness or spontaneous and ad hoc communication. RCSM is built on an object-based development framework, having application-specific adaptive object containers, and a context-sensitive object request broker. This middleware is used in a smart classroom. The Gaia project [25] at UIUC seeks to bring the same concept to computing. Gaia creates an environment to bridge the gap between virtual and physical objects. The goal of Gaia is to design and implement a middleware operating system that manages the resources contained in an active space. Gaia [26] is a component based distributed meta operating system that runs on existing operating systems and is implemented on top of existing middleware platforms. It includes the Unified Object Bus and the Gaia OS services. Georgia Institute of Technology has built an Aware Home [7]. This project design interactive experiences appropriate for people in an aware home environment, and address the software construction challenges of engineering a robust and reliable bridge between the designer's intent for interactive experience and the technology itself. They developed and refined the "Context Toolkit" [27] to support rapid prototyping of home applications that leverage knowledge sensed from the environment. Project Aura [28] is intended for pervasive computing environments involving wireless communication, wearable or handheld computers, and smart spaces. Aura provides a human distraction-free software platform. Aura spans every system level: from the hardware, through the operating system, to applications and end users. Aura applies two broad concepts. 1) It uses proactivity, which is a system layer's ability to anticipate requests from a higher layer. 2) Aura is self-tuning: layers adapt by observing the demands made on them and adjusting their performance and resource usage characteristics accordingly.

The project ArchWare [29] developed by Europe Unite aims to construct a evolvable system centered with architecture mode. ArchWare has proposed a dynamic architecture Description Language π-ADL [18] 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 [17] 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.

Compared with the above projects, our work focuses on the smart spaces for intelligent transportation system. We emphasize the semantic-integration and adaptive method for middleware platform, and have built the ScudWare middleware platform for pervasive computing systems. Different from the previous work, we present a ScudADL to describe the middleware dynamic adaptation, which has the dynamic re-configuration characteristic. The component-based run-time recomposition provides adaptability and scalability of the ScudWare. Moreover, we expend D-ADL and π-ADL and present a ScudADL framework with its specification semantics. It can describes dynamic entity structure characters, and dynamic behavior adaptation. Different from D-ADL and π-ADL, the computing resources consumption is concerned in ScudADL, and the component inner and outer adaptive behaviors are separated from component functional behaviors in an explicit way.

VII. Conclusions and Next Work

In this paper we propose a semantic and adaptive middleware platform called ScudWare for pervasive computing system in a smart CAA space. The ScudWare middleware is based on the adaptive communication environment and a extended TAO service, which includes adaptive components, and semantic virtual agents. The paper describes ScudWare component structure characters, and dynamic behavior adaptation by using a ScudADL framework and its specification semantics. The ScudADL

is based on higher-order typed π calculus theory. It achieves to separate component inner and outer adaptive behaviors from component functional behaviors in an explicit way. In addition, we introduce an application of ScudWare architecture, which is a computer aided assessment system in a smart CAA space, and give a case study to show its adaptation.

Tue further work includes 1) extension of ScudWare to other smart spaces; 2) development of more applications in smart CAA space using ScudWare platform.

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] Nigel Davies, Hans-Werner Gellersen, Beyond Prototypes: Challenges in Deploying Ubiquitous Systems, *IEEE Pervasive Computing*, pp.26-35, January-March 2002.
[3] Tim Kindberg, Armando Fox, System Software for Ubiquitous Computing, *IEEE Pervasive Computing*, pp.70-81, January-March 2002.
[4] Alastair Beresford, Csaba Kiss Kall, Ursula Kretschmer, Friedemann Mattern, and Martin Muehlenbrock, The First Summer School on Ubiquitous and Pervasive Computing, *IEEE Pervasive Computing*, pp.84-88, January-March 2003.
[5] MIT Media Lab, Smart Rooms, http://vismod.www.media.mit.edu/vismod/ demos/smartroom/
[6] MIT Media Lab, KidsRoom, http://vismod.www.media.mit.edu/vismod/ demos/kidsroom/
[7] Georgia Tech, Aware Home Project, http://www.cc.gatech.edu/fce/ahri/
[8] Margaret Fleck, Marcos Frid, Tim Kindberg, Eamonn OBrien-Strain, Rakhi Rajani, and Mirjana Spasojevic, From Informing to Remembering: Ubiquitous Systems in Interactive Museums, *IEEE Pervasive Computing*, pp.13-21, April-June 2002.
[9] Anand Tripathi, Next-Generation Middleware Systems Challenges Designing. *Communications of the ACM*, 45(6), pp. 39-42, 2002.
[10] 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.
[11] http://www.omg.org/technology/documents/formal/components.htm, 2005.
[12] http://www.cs.wustl.edu/ schmidt/ACE.html, 2005.
[13] http://www.cs.wustl.edu/ schmidt/TAO.html, 2005.
[14] 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.
[15] R.Milner, J.Parrow, D.Walker, A Calculus of Mobile Processes. Infromation and Computation, VOL.100, No.1, pp.1-40, 1992.
[16] D.Sangiorgi, Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD Thesis, University of Edinburgh, 1992.
[17] Li Changyun, Li Gansheng, he Pinjie, Formal Dynamic Architecture Description Language D-ADL, Journal of Sotware, VOL.17, NO.6, pp. 1349-1359, 2006.
[18] 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.
[19] Qing Wu, Inteligent CAA theory model and application, Master thesis, Hangzhou Dianzi University, 2003.
[20] Zhoubishui, Li jun and Wojunjun, Proof of Program Correctness based on Syntax Trees, Computer Applications and Software, VOL.24, NO.4, 2007.
[21] Zhou bishui, Zhang Yan-Hong, and Zhao Jing, Model and Algorithm Designing of the Proof of Program Correctness on XML Syntax Trees, Journal of Hangzhou Dianzi University. VOL.26, NO.1, 2006.
[22] Stanford Interactive Workspaces Project. http://iwork. stanford.edu/
[23] 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.
[24] 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.
[25] University of Illinois at Urbana-Champaign, Gaia Project, http://choices.cs.uiuc.edu/ActiveSpaces/
[26] Manuel Romn, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt, A Middleware Infrastructure for Active Spaces, *IEEE Pervasive Computing*, pp.74-83, October-December 2002.
[27] Daniel Salber, Anind K. Dey and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. *In proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 1999.
[28] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste, Project Aura: Toward Distraction-Free Pervasive Computing, *IEEE Pervasive Computing*, pp.22-31, April-June 2002
[29] F.Oquendo, B. Warboys, R.Morrison, et al. ARCHWARE: Architecting Evolvalble Software, EWSA 2004, LNCS 3047, pp. 257-271, 2004.

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.

Hu Weihua is a professor of computer science at Hangzhou Dianzi University. His major interests include Distributed Computing, Software Middleware, and CORBA Component Model, and CAA Theory.

Ding Wen is an associate professor at Zhejiang Education Examine. His major interests include CAA, Distributed Computing and Software Middleware.