Modeling and Description of Organization-Oriented Architecture

Munan Li
School of Business Administration, South China University of Technology, Guangzhou, China
Guangdong Province Key Laboratory of Innovation & Decision Management System, Guangzhou, China
Email: limn@scut.edu.cn

Junfeng Zhou and Xinyi Liang
School of Business Administration, South China University of Technology, Guangzhou, China
Email: {359226375, 605062514}@qq.com

***Abstract*—Traditional software architecture models, such as: Object-oriented, Service-oriented, Agent-oriented, have been confronting the challenges in more and more complex distributed computing situations, e.g. pervasive computing, Internet of things, smart-city, etc. Organization-oriented architecture model is proposed to attempt to improve the abstraction and design capability of software architecture in the pervasive computing environment, and then enhance the efficiency of architecture model reuse. The description language of organization-oriented architecture is based on the WRIGHT * and CSP.**

***Index terms*—Software Architecture, Pervasive Computing, WRIGHT, ADL**

I. Introduction

With the development of cloud computing and the pervasive computing technology, as well as some pervasive computing applications, the concept of traditional information systems has undergone some changes. Especially with the emergency of Internet of things, there is higher demand for the environment of pervasive computer because the space and information are sharply expanded by users [1]. Traditional information systems are used by human, therefore, an important design idea of information system architecture is needed to cover a basic utility paradigm: 1) how to improve the friendly "human-machine" dialogue to enhance the efficient use of information systems; 2) how to improve the reuse of code and architecture of software, so as to improve the production efficiency of information system.

For the technology promotion of Internet, smart sensor, multi-agent system etc, in some pervasive computing applications, human has no longer existed as pure user role, but become a part of the information system [2]. For example: in the medical care system located in the suburb of metropolis, the patients in need of medical care, and the elderly who cannot take care of themselves, have become parts of the information system. Information of these people, e.g. body temperature, blood pressure, heart signs, respiration etc, is an important part of the information system. In such human-machine symbiosis system, there are a variety of heterogeneous system and organization structure; system architecture modeling and the formal description of complex computing applications for this type of system, have a lack of enough relevant research [3][4]. But as can be expected, with the development of virtual reality, artificial intelligence and information technology, the applications of multi-agent system will become more and more popular because of its advanced sociality, adaptability, autonomy and mentality [5].

II. Comparisons Among The Main Architecture Model

The architecture model of software or information system is about the abstraction, modeling and description of the goal, vision and design specification of the system, which are used to guide the construction of the information system. When talking about service-oriented architecture, for instance, a manufacturing platform integrates different modules like the platform of management and maintenance, the self-designed system for industry and the e-business service system to provide better services which meet higher demand, it is difficult to integrate these systems because there are a great variety of term names, definitions and data format among them. Even with the development of network, it is possible for these applications to run on different computers and communicate between each other; there still remain problems such as network information congestions and inability of applications being online at the same time [6]. In the pervasive computing environment, such as: Smart-City, Internet of things, more complex cooperative solving system and so on, the granularity of traditional modeling element seems too small, so that it is difficult to provide the overall abstraction and modeling on architecture. As more macro and strategic aspects of the information system are concerned, for example: a remote health care system consists of multiple independent database application

doi:10.4304/jsw.9.4.867-872

system (E-government, E-medical system, hospital expert system, etc.), monitoring and sensor system (patient vital life-signs monitoring, indoor environment monitoring etc.), coordinated control system, induction calculation unit, communication control unit and so on, and involves many different types of the terminals, which have the different operating system, data format, control unit, even the protocol of communication. For the emergence of the problems of interaction, cooperation, communication and coordination, the traditional modeling elements: object, component, service, modeling element are incapable to cover the design problems of whole system, such as: interaction, collaboration, overall structure of composite event and service composite. Although the "Agent" can accommodate part of the interaction, coordination and response characteristics of the environment, but the existing Agent-oriented software engineering lacks of openness, especially the compatibility with the traditional computing unit and information application, the interoperability among multiple Agent platforms, and communication standardization of multi-agent system, etc [7].

III. Organization-oriented Philosophy

Attempt to be compatible with the current mainstream software architectures; "organization" is taken as one of basic modeling elements in a macro information system. Firstly, the element of organization is defined as the aggregation or assembly of Agents with different tasks; Secondly, the traditional "component" and "service" are the instantiations of organization in the framework of object-oriented programming. Although the Agent-oriented design pattern can meet part of the requirements on initiative, autonomy, collaboration, but there still exist such problems as: the lack of the standards among different agent-computing platform, the lack of interaction and fusion mechanism with the large number of non-Agent systems. Therefore, he Agent-oriented design pattern is unable to wholly satisfy the design requirements of pervasive computing, distributed computing, smart-city, etc [7, 8].

Recently, organization-oriented philosophy began to get more attentions. The discussion on the openness of organization computing between Olivier and Fred [9], the organization computational models provided by Huib and Virginia based on Organization Theory [10] and the organization theory that Esther and Chimay attempted to make use of in construction of supply chain [11] are all typical examples.

Organization-oriented architecture is proposed to improve the capabilities of abstraction and modeling on more complex and huger information system, such as: smart city, Internet of things, smart sensor network, etc. Currently, the mainstream architecture models seem difficult to cover the overall design of such giant information system, so that the cooperation, synchronization and coordination among a large number of independent computing units and applications cannot be integrated efficiently, and the risk of giant information system will increase. In the organization-oriented architecture, "organization" is defined as the facet, meta-element of modeling architecture, which is compatible with non-agent systems including object-oriented systems, service-oriented applications.

The current mainstream architectural patterns are generally described as triples: <components, connectors, architecture configuration>, and now most of the architecture description languages are expanded around the components [12].

Although collaboration and synchronization are in the scope of object-oriented thinking, from the perspective of designing complex distributed collaboration systems, the particle sizes of classes and components are still small. Some researches might think the components (services) equal to the agents, but the biggest difference lies not in whether it can move (such as: object-oriented concept of the moving objects), but in targeting and specific modeling. Components are the results of code reuse, which are used to improve the scalability and maintainability of software systems. Essentially agents can be considered as the package and reuse of responsibilities and objectives, direct goals and mission-oriented. Although a single agent only has limited computing power, but complex functions and objectives can be completed through consultation, coordination and collaboration.

In general, a typical component is a business logic package, focusing on the generic function package, such as EJB and COM+, web services and SOAP, cross-platform protocols are derived for communication between these heterogeneous components, as the results of packaging the business functional components. Obviously, service can be combined into components with larger particle size. But compared with agent, service lacks autonomy and social characteristics, such as initiative, goal-oriented, and collaborative, and it hardly can fully characterize the high-level modeling and design requirements of complex distributed system.

The typical agent is a package of tasks and objectives, self-governing body with some smart features of the (Belief, Desire, Intention, and Motivation) [13, 14], for example, spiders for Internet searching, simple tasks performed by mars exploration robots. In general, agent has "strong" definition and "weak" definition [13]. Here is a narrow definition: agent is a computer software and hardware entity with roles and responsibilities, collaborative willingness and ability, and limited computing power.

Definition 1: Agent is a triple [14, 15]:

$Agent :=< Res, \Omega, \Phi >$

$Res :=\{< Role><Responsibility>\}^*$

$\Omega :=\{< Computation><Reasoning><Cooperation>\}^*$

$\Phi :=\{< Belief><Desire><Intention><Motivation>\}^*$

Res is the set roles and responsibilities, Ω is the set of ability, including collaborating, calculating and reasoning ability, and Φ is a collection of psychological states. And here the traditional BDI structure is chosen as the psychological state.

In reference [16], collaboration is defined as allocating tasks, information and resources among agents working together, which are aimed at enhancing the viability, improve the performance and conflict resolution of the system. We put Organization as the spontaneous aggregation of agents, and the collection of behavior taken by the collaborating entities is for a common goal.

Definition 2: Organization is combined by agents and other functions or service units, is a triple.

Organization :=< G, A, R, E>

*G :={< Goal>}**

*A :={< Agent>}**

*R :={< Relation><Behavior>}**

*E :={< Constraint><Evaluation>}**

G is the set of organization missions and goals, A is the collection of agents, R is the set of behaviors and relations, and E is the environmental constraints and interactions, including the function and service unit of non-agent part. Here the organization is considered as collective aggregation of multiple agents, by behavior planning, reasoning based on the BDI, completing the assigned task (target), and interacting with the constrained environment. Wooldridge and Jennings [13] formally described the collaborative process of the main body by using quantitative multi-modal logic. They proposed that agent collaboration process includes four phases: finding potential synergy, forming a team, building plans and taking team action. From Definition 2, O can be considered as a prerequisite to find potential synergy, and the common tasks and goals lead to the possible collaboration. After reasoning the possibility of collaboration, an organization based on agent is generated, this is A, the set of agents. After the organization team, behavior queue B will be produced through local and global planning. And, the interaction with the constraint environment will come after taking action. Besides the participating agents, organization collaborative tasks also require third-party collaborative supporting mechanisms, for example, environment for agent migration, arbitration for collaborating conflict resolution, discharge of process resource contention, maintenance of sharing information and knowledge sources, so organization-oriented applications need to consider the collaborative support mechanisms. A formalized definition of cooperation/collaboration support is given in this paper.

Definition 3: Cooperation support is the service provider of the mechanisms and services, communication, arbitration, coordination, etc, required by organization collaboration. Collaborative support can simply be attributed to a triple [15]:

Cooperation_support := <Services, Customers, Constraints>

*Services := {<Port><Content>}**

*Customers := {<Agent>}**

*Constraints := {<Rule><Specification>}**

In IEEE1471, passed in 2000, architecture is defined as the basic structure of a system, including the various components, the mutual relations, and the relationship with the environment, design guidelines and evolution principles. Here we propose an architecture model for collaboration-oriented system [16].

Definition 4: The architecture model for organization-oriented system is defined as:

SA_OgM := {Organization, Supporters, Configurations}

The organization is the design element, as defined in Definition 2. And supporter is the collaboration support unit, providing mechanisms and services needed for collaboration, such as communication, interaction protocol analysis and coordination. Configuration means the architecture configuration, including environmental constraints and complex rules, and can support more advanced abstraction and reuse, like architecture style. Essentially, SA_OgM is based on agent theory and methods, and it is the generalized package of agents.

This package is conducive to design application systems where collaboration is the main form, for example, pervasive computing applications, virtual manufacturing, operational command simulation, and other distributed collaborative applications. From the practical application viewpoint, it can achieve the smooth transition from requirements process to architecture design process by taking organization as the main description and design elements of this type of system. After clarifying requirements specification, the task and role of the system and the participating body can be organized together organically, and the domain analysis and system analysis can be separated maximally, thus the domain experts can participate to complete software system design genuinely.

IV. Organization-oriented Architecture Description and Description Language

Currently, the architecture description is divided into non-formalization and formalization. Standard IEEE 1471 defined the architecture and the architecture description as two different concepts [17]. As to non-formalized description, block diagram as represented, there is still a relatively large gap in the amount of information provided, compared with formalized description, like ADL. Today, architecture design is mainly based on ADL, and the codes and presentation systems are generated from the formalized results. In the research of architecture, there have been some excessive expectations and requirements trying to establish a unified framework which is versatile, easy to understand, adaptable to environmental changes, and even able to generate systems that can run directly.

Essentially, architecture is the embodiment of system design experience and knowledge, both the architectural pattern and style. In the practical project environment, if the experience and knowledge of experts can be well transferred and reused, the efficiency of system design and development can be greatly improved. And the efficiency of software system development can be enhanced significantly by providing consistent and comprehensive supporting tools and platforms for architecture design. In essence, architecture wants to

achieve reuse in the highest level of system design. So, what's the fact of reuse? This question has been ambiguous; actually, what is to be reused is the design experience and knowledge. Therefore, the architecture is thought to be the carrier of design experience and knowledge. And through extraction and reuse of the design experience, the highest level of reuse can be achieved in the software engineering field, which improves the efficiency and success rate of the complex systems construction. Higher level of abstraction modeling is needed to extract design experience and knowledge, for example, using ontology to express knowledge is an intuitive and composite knowledge modeling method. In fact, as one of the modeling elements, collaboration implies the experience and knowledge for a successful collaboration unit, so the experience and knowledge contained in the architecture designing can be retained and delivered more completely.

A. Description of Organization-Oriented Architecture

The formalized description should be considered when proposing and researching the architecture. In the standard IEEE1471, mentioned above, there are no inevitable relations between architecture and components, but, the mature architecture description languages are now based on components commonly. Architecture description languages are generally sorted to Implementation Independent Language and Implementation Constraining Language.

The representative languages for IIL include such traditional framework as: WRGHT [18, 19], Rapide, Unicon and MetaH for ICL. Rigorous presentation logic and operation rule for syntax and semantic are necessary for architecture description language. A mature and completed architecture description language should support the description of the architecture configuration, the description of the architecture style, property analysis and practical application.

Here, WRIGHT is chosen as the architecture description language of collaboration-oriented system, which is one of the implementation independent languages. The traditional WRIGHT is good at formalized description and design for distributed systems, but lack timestamp describing mechanism. Under the pervasive computing environment, most of the events are timestamp-based timing events, so appropriate extension is necessary based on WRIGHT. The formalized description, by using time extended CSP in the basic semantic model, is to correspondingly expand the semantics operating part of WRIGHT by TCSP [20]. In order to distinguish it from the original WRIGHT, this new language is called as WRIGHT*, which is based on CSP semantic operation, with some grammar expansion.

(1)SpecList := Spec | SpecList Spec ;
(2)Spec := Configuration | Style ;
(3)Style := "Style"Name
TypeList
"Constraints"
ConstraintList
"End Style";
(4)TypeList := Type | TypeList Type | null ;
(5)Type := Component | Connector | InterfaceType | Organization | Supporter | TaskType | AgentType ;
(6)Component := "Component "
SimpleName ['(' FormalParams ') '] PortList
"Computation = "ProcessDescription ;
(7)Connector :="Connector"
SimpleName ['(' FormalParams') ']
RoleList
"Glue = "ProcessDescription ;
(8) Organization := "Organization"
SimpleName ['(' FormalParams ') ']
TaskList
AgentList
"Behavior = "ProcessDescription
(9)Supporter := "Supporter"
SimpleName ['(' FormalParams ') ']
ServiceList
CustomerList
"Support="ProcessDescritpion
(10)ConstraintList : = Constraint Expression | ConstraintList ; Constraint Expression | null ;
(11)Configuration :="Configuration"Name
"Style"Name
TypeList
"Instances"
InstanceList
"Attachments"
AttachmentList
"End Configuration";
(12)InstanceList := Instance | InstanceList Instance | null ;

From the description of grammar above, there is just a little number of expansions in order to keep the original grammar structure of WRIGHT: adding Organization and Supporter in the Architecture modeling elements, and the syntax description of these two elements refers to the Definition 2 and Definition 3. In order to better describe the unified pattern of behavior in the style description, the meaning of *InterfaceType* is expanded to support properties like Task and Service. Besides, Set operators related to Organization and Supporter need to be added, such as Tasks(c), Agents(c), Services(s), and Customers (s).

In the description of "***Organization***", Line 5, Agent is put as broad, intelligent, autonomous, pro-active components, although theoretically, here we can use Component instead. In order to keep the semantics difference and the integrity of the original formal language, here the Agent is a default meta-element, just as the concept of object of object-oriented is implied in component / service.

In the process algebra (such as: CSP, PEPA, and CCS, etc.), process is always described as an entity and as an agent in some cases, therefore, there are collaborative operators in process algebra [18, 19]. So the original synchronization operations have been expanded to support collaboration in ordinary sense. The collaboration here can be understood as a type of Super-event or Composite event. In order to describe the collaborations

between processes, collaborative operator Θ is introduced into CSP representing Organization collaboration services between Agents (Processes).

Definition 5: Organization collaboration is an event composition, which means multiple agents taking cooperative actions for the common goal of the system.

Let A is the collection of Agent or processes; Event-coop= $\forall A_i, A_j \in A|\ i, j = 1 \ldots n, i \neq j, (A_i \Theta A_j)$ is a composite process event; and the collaborative operator Θ has the following properties:

Atomicity. Same as other common event, a collaborative event is an atomic event, which is an integration that cannot be divided into two or several fragments.

Terminable. If i=j, then $(A_i \Theta A_j) = (A_i \| A_j) \square \surd$, where $A_i, A_j \in A$

Symmetry. $\forall A_i, A_j \in A|\ i, j = 1 \ldots n, i \neq j, (A_i \Theta A_j) = (A_j \Theta A_i)$

Inner Commutative –law. $\forall A_i, A_j, A_l \in A|\ i, j, l = 1 \ldots n, i \neq j \neq l, (A_i \Theta A_j \Theta A_l) = (A_i \Theta A_l \Theta A_j)$.

Goal-consistency. $\forall A_i, A_j, A_k \in A \mid i, j, k = 1 \ldots n, i \neq j \neq k$. If event $A_i \Theta A_j$ has the same goal as $A_j \Theta A_k$, then $(A_i \Theta A_j) \cong (A_j \Theta A_k)$, which means $A_i \Theta A_j$ is equal to $A_j \Theta A_k$.

Theorem1. $\forall A_i, A_j, A_k \in A|i, j, k = 1 \ldots n, i \neq j \neq k$. $((A_j \Theta A_j) \| A_k) = ((A_j \Theta A_i) \| A_k)$

Proof: Clearly, it can be proved by the properties: Atomicity and Symmetry.

Theorem2. $\forall A_i, A_j, A_k, A_l \in A|i, j, k, l = 1 \ldots n, i \neq j \neq k \neq l$, If$(A_i \Theta A_j)$ is equal to $(A_j \Theta A_k)$, then $(A_i \Theta A_j) \| A_l$ is equal to $(A_j \Theta A_k) \|_{Al}$.

Proof: It can be derived from Definition 1 and Definition 5. This theorem means that there are two separated collaboration events with a consistent goal, and they have same effect on the next event of the system. But this theorem is irreversible, and you cannot derive events equivalent from the same effect. Goal congruence lays the foundation for higher-level events composition.

Theorem3. $\forall A_i, A_j, A_l \in A|i, j, l = 1 \ldots n, i \neq j \neq l, (A_i \Theta A_j) \Theta A_l \neq A_i \Theta (A_j \Theta A_l)$.

Proof: Apparently, this can be proved from the atomicity of the collaboration operator Θ. And theorem3 means that process composited or aggregated from several atomic events is specific goal-driven, which is not conflicted with Property 4. Here is a simple example: in a remote medical diagnostic system, which means the collaboration between temperature detection event Ai and blood test event A_j is necessary for a detailed laboratory and testing report, in which maybe A_i is just a remote data transmission. Generally speaking, it is more complete and efficient to transport the summary of test results than data of each detected event combined with A_i. In fact, when the doctor in the remote terminal only gets the testing data about some special parts of the body, maybe he cannot do anything.

B. Organization-Oriented Architecture Style

Sometimes, it is possible for the system designers to care more about the style of software architecture. One style of architecture defines that the configurations can share the attribute set of the information systems [21]. As the granularity of model element in organization-oriented modeling is bigger than the traditional, it can reduce the complexity of architecture description. A simple description of the "organization–supporter" style is:

Style Organization- Supporter
Organization Org (nt: 1…; na: 1…)
Task task1…nt
Agent agent1…na
Supporter Supp (ns: 1…; nc: 1…)
Service Serv1…ns
Customer cus1…nc

Interface Type taskType = (read? x→$\overline{recognize?x}$ → taskType ∏ read? x→$\overline{refuse!x}$ →taskType) (close→§);

Interface Type agentType = (get?x →$\overline{response!x}$ → agentType)∏get?x →$\overline{refuse!x}$ →agentType ∏$\overline{request?x}$ →agentType) (idle→§);

Interface Type serviceType= (request?x→($\overline{serviceUp?x}$ →servieType ∏$\overline{refuse!x}$ →serviceType)) close→§;

Interface Type customerType= (register?x→($\overline{accept?x}$ →customerType ∏$\overline{reject?x}$ →customerType)) close→§.

Constraints

∀c: Organization • Type(c) = Org; ∀i:1...nt • t_i:Task| t_i ∈ Tasks(c)•Type(t_i) = taskType; ∀j:1...na • a_j:Agent| a_j ∈Agents(c) = agentType • Type(a_j) = agentType ∧ ∀s:Supporter • Type(s) = Supp; ∀k:1...ns • se_k:Service| se_k ∈Services(s) • Type(se_k) = serviceType; ∀m: ...nc • cum: Customer| cum ∈Customers(s) • Type(cum) = customerType.

WRIGHT language is taken as the ADL blueprint, for the consideration to the high-level abstraction ability of WRIGHT, and the scalability of CSP is so basic that it can simply describe the architecture of organization-oriented system. As the CSP does not support the factor of time, some scholars put forward time based on an extension of CSP [19, 20], but that the interaction has the time characteristic in the formulation of the problem still exists.

V. CONCLUSION

With the development of computing system and the related theory, especially, such as the cloud computing, pervasive computing, smart city, and service-supermarket etc, we advanced a novel modeling of computing architecture whose meta-element is the "***Organization***". Through justified expanding of the traditional WRIGTH language, this novel model of computing architecture can be described rationally. From the development of the software and computing architecture, the popular models of architecture are difficult to solve those problems about more and more complex applications of pervasive computing. In essence, the traditional object-oriented methodology is not intuitionistic or efficient enough. Therefore, the novel element of modeling on the software

architecture may be a valuable attempt on the modeling of pervasive computing.

As for the traditional architecture models based on the component-oriented or object-oriented method, it is designed to encapsulate the function and the reuse of software code. But, in the pervasive computing systems (Cloud-Computing, Smart City, Internet of Things), the organization, cooperation, interaction, rationality of agent and adaption of software agents are all the important goals of architecture designing, and the Organization (just like the component to the object-oriented method) is the aggregation of agents, which is the solution to the real requirement of application. Therefore, the Organization is the reuse of solution, which is a higher level reuse with the task solving and the computing entity.

About the formal description of the novel architecture-COA, we choose the WRIGHT as the cornerstone, which is irrelevant to the implementation, and supports the higher level modeling of requirement. Through the justified expanding of the traditional WRIGHT, we provided a framework of architecture description. But the behavior modeling of WRIGHT is based on the CSP, which is limited in the time sequence or time sensible events. Therefore, the novel modeling element-Organization and WRIGHT* should be further discussed. In our future works, we will focus on such aspects as: formal description of Organization-oriented architecture, the design philosophy of Organization-oriented system, the style of architecture, the design pattern of Organization-oriented system etc.

ACKNOWLEDGEMENT

This work was supported in part by a grant from Open Foundation of Guangdong Province Key Lab (No.2011A060901001), Fundamental Research Funds for the National Universities of China (No. 2013XMS03), Major Program of the National Social Science Fund (No. 11&ZD154).

REFERENCE

[1] Benjun Guo, Jianhong Gan, Jingwen Su, Yuewen Hu, Jun Lu. A Pervasive Computing Model of Internet of Things based on Computing Area Network. Journal of Computers, 2012, vol. 7, no. 7, pp. 1647-1654.

[2] ZHOU Chunjie, MENG Xiaofeng. The Researches and Challenges of Complex Event Detection in Pervasive Computing. Journal of frontiers of computer science and technology, 2010, vol. 4, no. 12, pp. 1057-1071.

[3] JIA Chaoguang,WU Qing,WAN Jian. Research on Dynamic Evolution Software Model for Pervasive Computing. Journal of Hangzhou Dianzi University, 2010, vol. 30, no. 1, pp. 38-41.

[4] TANG Lei, ZHOU Xingshe, YU Zhiwen, et al. Research and Implementation of Adaptive Entity under Ubiquitous Computing Environment. Journal of Xi'an JiaoTong University, 2011, vol. 45, no. 2, pp. 102-106.

[5] Weidong Zhao, Haifeng Wu, Weihui Dai, Xuan Li, Fei Yu, Chen Xu. Multi-agent Middleware for the Integration of Mobile Supply Chain. Journal of Computers, 2011, vol. 6, no. 7, pp. 1469-1476.

[6] Pengshou Xie, Zhiyuan Rui. Study on the Integration Framework and Reliable Information Transmission of Manufacturing Integrated Services Platform. Journal of Computers, 2013, vol. 8, no. 1, pp. 146-154.

[7] Shoham Y. Agent-oriented programming. Artificial Intelligence, 1993, 60(3):51-92.

[8] MEI Hong, SHEN Jun-Rong. Progress of Research on Software Architecture. Journal of Software, 2006, vol. 17, no. 6, pp. 1257-1275.

[9] Boissier Olivie, Hübner Jomi Fred, Sichman Jaime Simo. Organization oriented programming: From closed to open organizations. Lecture Notes in Computer Science, 2007, vol. 4457, pp.86-105.

[10] Aldewereld Huib, Dignum Virginia. OperettA: Organization-oriented development environment. Lecture Notes in Computer Science, 2011, vol. 6822, pp. 1-18.

[11] Bonyo Esther A, Anumba Chimay J. Organization-oriented multi-agent systems for construction supply chains. Electronic Journal of Information Technology in Construction, 2011, vol.16, pp. 727-744.

[12] LING Xiaodong. A Review of SOA. Computer applications and software, 2007, vol. 24, no. 10, pp. 122-124.

[13] Wooldridge M, Jennings N. Intelligent Agents: Theory and Practice. The knowledge engineering review, 1995, vol. 10, no. 2, pp. 115-152.

[14] LI Munan PENG Hong, LI XiangYu, et al, A Novel Modeling Architecture of Agent. ACTA AUTOMATICA SINICA, 2007, vol. 33, no. 1, pp. 15-20.

[15] Munan LI, Hong PENG, Jin-song HU. Research on Modeling and Description of Software Architecture of Cooperation-Oriented System. Lecture Notes in Artificial Intelligence (LNAI), 2006, vol. 4088–0546, no. 8, pp: 546-551.

[16] Munan LI, Junxia XIONG. A Novel Method of Model Composite in Decision Support System. Advances in information sciences and service sciences, 2012, vol. 4, no. 15, pp. 317-324.

[17] IEEE ARG. IEEE's Recommended Practice for Architectural Description, 2000, IEEE P1471-2000.

[18] CUI Xiaole, ZHANG Xing, MIN Jun, et al. An Extension of Wright for Real-Time Software Applications. Microelectronics and computer, 2006, vol. 23, no. 3, pp. 11-15.

[19] Medvidovic N, Taylor RN. A Classification and Comparison Framework for Software Architecture Description Language. IEEE Transactions on Software Engineering, 2000, vol. 26, no. 1, pp. 70-93.

[20] Song Jinjing, Shen Jun. Description and Simulation of Network Protocol based on CSP. Journal of Southeast University (Natural science edition), 2008, vol.38, no. Sup (I), pp.29-33

[21] Ma Jun-tao, Fu Shao-yong, Liu,Ji-ren, A-ADL: an ADL for multi-agent system. Journal of Software, 2000, vol. 11, no.10, pp. 1382-1389.