Research for an Intelligent Component-Oriented Software Development Approaches*

Youtian Qu
College of Information Science & Engineering, Zhejiang Normal University, inhua, 321004, P.R. China,
quyt@zjnu.cn

Chaonan Wang, Lili Zhong, Huilai Zou and Hua Liu
College of Information Science & Engineering, Zhejiang Normal University, inhua, 321004, P.R. China,
Email: {wcn, 04600110, 2008210785, liuhua} @zjnu.net

***Abstract*—Using software agents as next generation flexible components and applying reuse technologies to rapidly construct agents and agent systems have great promise to improve application and system construction. The increasing complexity of software has made it necessary to reuse software. Reuse has increased the reliability of software applications and made it efficient to develop and maintain current software. An Intelligent component-oriented software development approach, which emphasizes the design and construction of software systems by using reusable components, is an effective approach to the software development. Combining the advantages of agent-oriented and component-oriented methods, it aims to create more flexible, reusable and customizable agent components in future. An agent component-based architecture is proposed and a concrete application system is described to illustrate the method and process of applying the architecture**

***Key words*—Intelligent component, Agent Component, MAS, Software Reuse**

I. INTRODUCTION

The increasing complexity of software has made it necessary to reuse software. Reuse has increased the reliability of software applications and made it efficient to develop and maintain current software [1]. Software development needs to progress from handcrafted, line-at-a-time techniques to methodologies that support reuse of existing software assets [2]. In order to cope with this complexity, we need new paradigms to facilitate the design of distributed and open systems. Traditional software engineering methodologies are giving way to new software development paradigms. Component-oriented software engineering and agent-oriented software engineering are two paradigms that are garnering attention. Both of them seem to represent the evolution of the object-oriented paradigm. However, each paradigm focuses on different aspects of distributed applications and so they might be complementary. In the future, passive software components will be liberated by the proactive and social nature of agents. In effect agent-based technologies provide the mechanism for components to seek work, enter into cooperative agreements and thus otherwise address the requirements of dynamic, heterogeneous environments.

Agent technology is a complex software technology. To make this technology easier to understand, we need to encapsulate the complex concepts of agent technology in order to disburden the development of agent systems. We find the component technology is considered to be more suitable for distributed system development due to its granularity and reusability, and it is also a suitable means to disburden agent software development by making certain agent concepts customizable and reusable [3]. The Agent-based programming is emerging as a new programming paradigm in the next decades. However, there have been no programming languages to well support agent programming naturally. To make it worse, even before we try to implement agents, there is no proper modeling methodology for agent software development. The existing software development methods such as structured modeling and object-oriented modeling are not well suitable for agent software development because of the difference of notions in different paradigms.

Although the flexibility of agent interactions has many advantages as well as disadvantages which lead to unpredictability in the run-time system when engineering systems become complicated. On one side, decisions about the number, pattern and timing of interactions depend on a complex interplay of the agent's internal state and the agent's perception of the environmental and organizational context that exists when the decision is made. It is difficult for the agent to make predictions about the system's interactions. On the other side, there is a de-coupling, and potentially a considerable degree of variability between what one agent requests through an interaction initially and the outcome that eventually ensues. The request may be immediately accepted, refused completely or modified through some form of social interchange [18].

* Founding information: This work is partially sponsored by the Natural Science Foundation of Zhejiang Province, China (M603245, Y106469), and National high tech research and development plan (863plan):2007AA01Z105-05

© 2009 ACADEMY PUBLISHER
doi:10.4304/jsw.4.10.1136-1144

All in all, to the agent based software development process, there are still some issues for us to resolve. First, the issues of performance and efficiency based on agent software development have not been properly settled which have been baffling this field for a long time. Second, it lacks a kind of combination among the mature models of development, methods and tools to describe the agents-based analysis and design process. Third, the "dynamic" and "continually scalable" features in the process of system development have not been implemented.

Because the components such as CORBA, DCOM, JavaRMI are all based on classic Client/Server models, they have some pitfalls in autonomy, flexibility, initiative and network environment adaptability. Meanwhile, because the components determine their connectivity mechanism prematurely, to some extent, their "dynamic" and "scalable" features are affected and constrained.

As the autonomous software entities, agents represent potential components in a software system. Therefore we try to use a method combined with agent technique and component technique to build an agent component (AC). The method should offset the drawbacks of the single agent or component development method and meanwhile it can take full advantage of the features of agent technique and component technique.

The rest of this paper is organized as follows: The motivations and methods of combining agent-oriented and component-oriented are introduced in Sect. 2. The agent component-based architecture, the agent component-oriented software construction methods, the definition of agent component, and the agent component-oriented MAS general model framework are given out in Sect. 3. A model application instance and a concrete software application example are given out in Sect. 4. Finally, a simple summarize is given out in Sect. 5.

II. AGENT/INTELLIGENT COMPONENT OUTLINE

Software agents offer great promise to build loosely-coupled, dynamically adaptive systems on increasingly pervasive message-based middleware. Agents are specialized kinds of distributed components, offering greater flexibility than traditional components [4]

A. Motivations

The main difference between agents and components is the mechanism they use for communication. Agents use ACLs(Agent Communication Languages), while components use a metaobject protocol [5].

In the paper [5], Bergenti has given out the comparison between agents and components starts from table 1. It shows some important component-oriented abstractions and associates them with agent-oriented counterparts. Table 1 compares some of the abstractions that form the components' Meta-model with the corresponding abstractions of the agents' Meta-model.

The different communication mechanisms influence how agents and components open themselves to the outer world. Components use interfaces to enumerate the services they provide and to tell clients how to get in contact with them. The agent-oriented approach eliminates interfaces and provides agents with capabilities of describing what the agent can do, and how the agent can interact with other agents, i.e., the interaction rules it can adopt. By similar comparison in [5,6,7,8], the two major results can be concluded: 1) agents are more reusable and more compositive than components, and 2) agents allow to describe systems at a higher level of abstractions than components.

TABLE I. COMPARISON BETWEEN COMPONENT-ORIENTED AND AGENT-ORIENTED ABSTRACTION

Abstraction	Component-Oriented	Agent-Oriented
Communication	Task delegation	Task and goal delegation
Message	Requests for action	ACL message
Interaction with the environment	Events	Updates of believe
State	Properties and relations	Mental attitudes
Interactions	Interfaces	Capabilities
Between parties	Interface repository	Semantic matchmaker
Runtime	Application server	FIPA platform

Though agent has so many advantages, as we mentioned in section 1, agent-oriented software development is short of supporting of methods, tools, etc. Currently, CBSE (Component Based Software Engineering) has been a matured development method, and it has more technical supports, correspondingly. Therefore, we take the agent component as the next generation component [9].

B. Combining Agents and Components

Many similarities can be found between agent technology and component technology [8]. But we are more interested in the complementary concepts of these technologies, because these are the concepts that make the profit out of combining agents and components. We focus on the main purpose of agents in the communication ability, which gives us the possibility to process complex tasks by assigning single task to different agents. Instead, components focus more on reusability and parametrisation / customization aspects for the deployment of a component in different contexts. Fig.1a illustrates the process of combing components and agents, and Fig.1b illustrates the concept structure. The agent component (AC) is comprised of three parts: component, agent, and connector (Combines).

Component. Component is an independent functional entity in a system. In general, a component possesses its interface specification and internal specification by itself. Interface specification is mostly to describe the message provided to the user by the component. It can be divided into two sorts: The first is the functional specification, which is the exterior user interface provided by the component; the second is the entry point, which is the exterior interface used by a component. Internal specification mostly includes the syntax restriction of the component architecture of itself (e.g. the dependence of the functional specification and entry point, namely those

© 2009 ACADEMY PUBLISHER

exterior interfaces which support a certain functional demand), semantic models (e.g. the interaction protocol with the other components) and the other service features (e.g. thruput etc.) [10]. In the agent component, it presents as a service entity, namely a service component.

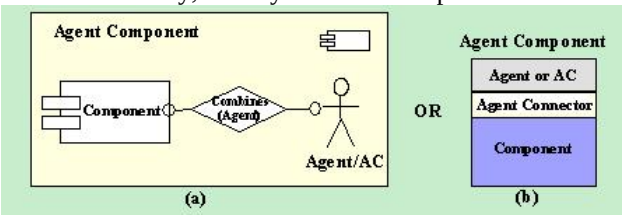

Figure 1. Combines Agents and Components

Agent. The major function of the agent in the model is to realize the communication based on MAS. It may be not a concrete computing entity, but a communication entity. In essence, it can be a concrete computing entity completely when it needs to provide some particular function demands (e.g. needs to provide some intelligent, automatic services). Here, the agent has not only the ability of communication but also the service ability of supporting for the outer world. Therefore, in essence, the agent in the model is an AC. It has both the component's services ability and the agent's communication ability. We adopt the particular style BDI (Believe, Desire, Intention) model system to design and implement the agent.

Combines. The combining part is a core part in the AC, it combines the basic features of agents and components, and it is the token entity, which is to realize the communication with the other ACs and to provide the services for the outer world. It combines the communication ability of agent, the interface specification of component, the internal specification and the service features to endow itself with the service ability and to provide or request service to the outer world. The concrete implementation of service is achieved via the BDI model based on agent. The combining parts in Fig.1 are designed and realized by agent. Therefore, in modality, the combining part is taken as a connector, which is realized by agent. The combining part is a gluing body, by gluing the agent and component, it conceals the interfaces, properties and relations which the component directly provides to the outer world. Substituting the endowed service ability by the component for the inherited communication ability by the agent, It serves as a service component to support the assemble service for application system development, and it also serves as an agent to support the dynamic interaction with the other agent components in the runtime.

After examining its basic properties, it is possible to create a single, comprehensive definition of an AC: An AC is a language-neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published service interfaces (service ability of AC). An AC is not platform-constrained or application-bound. An AC should be autonomous, reactive, pro-active and has social competence [11].

An AC has communication abilities (the agent's features) and it can be reused and parametrized (the included component features) for different contexts. In the Fig.1, if we take the "Agent" as a component or an agent component and take the "combines" as an agent or an agent connector, the flexible and scalable software architecture can be achieved. With our approach we want to disburden agent software development. The component technology is a suitable instrument for this task by making certain agent concepts customizable and reusable. Here "reusable" means to have an AC that can be instantiated for every agent we need. And "customizable" means to connect AC instances, add/remove certain ontologies and add/remove behaviours graphically. In general, the AC has all agent properties like autonomy, reactivity, proactivity and interaction and adds component features like reusability and customizability. An AC is a generic component that describes general services that every agent must provide and so an AC can be instantiated/ reused for every agent one wants to build.

Combining agents and components into reusable software components provides several advantages:1) Applications are developed by selecting and assembling the appropriate components. 2) Integration and inter-operability among agents, standard component-ware technology and supporting tools are assured. 3) Developers who are unfamiliar with agent concepts and notations may choose agent-based components to fulfil specific functionality of their applications. This enables agent technology to be easily assimilated into current engineering practice[6]. Anyway, it is worth noting that the very naive approach of encapsulating an agent into a component so as to run it in an application server has some drawbacks. The most remarkable one is that the threading model that the application server imposes to components may not be compatible with agents. Basically, the developer must choose between loosing some enterprise feature, e.g., fault tolerance and transparent scalability, and implementing only reactive agents. Some middleware providing a reasonable compromise are already available now, e.g., Caribbean and EJB 2.0 allow asynchronous messaging in EJBs [5].

III. THE AC-BASED ARCHITECTURE AND THE AC-ORIENTED MAS MODEL

Developing good software architecture for a complex system is a critically important step for insuring that the system will satisfy its principal objectives [12]. The software architecture is widely recognized as one of the most fundamental concepts in software engineering, because of the fact, that today's software systems are assembled from components with different characteristics: for example, heterogenous, legacy or distributed systems. At the software architecture level, designers combine subsystems into complete systems using different techniques [13]. The architectural design plays a key role in software engineering. The software architecture is the backbone of the design solution, it has the functional

© 2009 ACADEMY PUBLISHER

requirements of the system and satisfies the quality requirements[14]. The concrete implementation of service is achieved via the BDI model based on agents (Fig.2).

Agent Component. The "Agent Component" is composed of "component" and agent or agent component. This component represents the component features in the deployment of design or in the assembly of components, and in the runtime, it represents as a component-based agent, which possesses all characteristics of an agent.

Connector(Agent). Connector was introduced in software architectures as a first-class entity representing component interactions. The basic idea is that application components contain only the application business logic, leaving the component interaction- specific tasks to connectors. However, such a characterization is too vague, since it does not strictly draw a line between component and connector responsibilities [15,16]. The connectors described in Fig.2 are realized by agents, which take full advantage of the ability of the communication interaction of agents, and use the apperceiving and message communication mechanism of agents to complete the assembly combination of components. In the proposed architecture, connectors coordinate protocol execution. Connectors accept and interpret protocol descriptions at run-time. Therefore interaction protocols do not need to be pre-coded in the connectors. Protocol descriptions include the definition of the messages exchanged, as well as the internal actions carried out by the agent during protocol execution. The interaction protocol is linked or connected in this way with the agent's functionality. In essence, the architecture style is similar to the architecture style of message bus-based or message router-based.

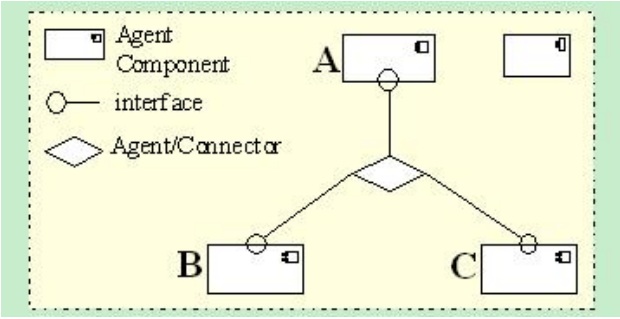

Figure 2. Agent Component Based Architecture

In the design and implementation, using the communication mechanism of agent to achieve the function of connector makes the architecture more flexible, reusable, configurable and combinable.

Agent Component Assemble. Each A, B, C is presented as an AC in Fig.2. In essence, an AC looks like a subsystem. It may be realized by combining agents and components. It may be realized by combining agents and ACs.

Communication Style. The communication style introduced in this paper is based on MAS communication mechanism, and realized by multi-agent.

Especially, in the architecture, if the agent connector which combines A, B and C is taken as a MAS, and then a general agent component-oriented MAS model can be achieved (Fig.3). In Fig.3, all components/agent components are out of the dashed circle and all the connectors are in the dashed circle. As we have mentioned above, the connector is designed as an agent, and all of the agents make up of a multi-agent circumstance, thus we can take those agents in the dashed circle as a MAS. In the center of the circle, there is a virtual agent /social agent which realizes the communication between agents (connectors). Each connector has concealed the features of a component instead of the service ability of an agent component. This model indicates that an agent component can present an application service, an application system or subsystem. An application system is composite of MAS (or agents), components, and application subsystems (ACs). Therefore, the result of the model applying is also an AC. It makes the software development process easier to the ordinary development methods.

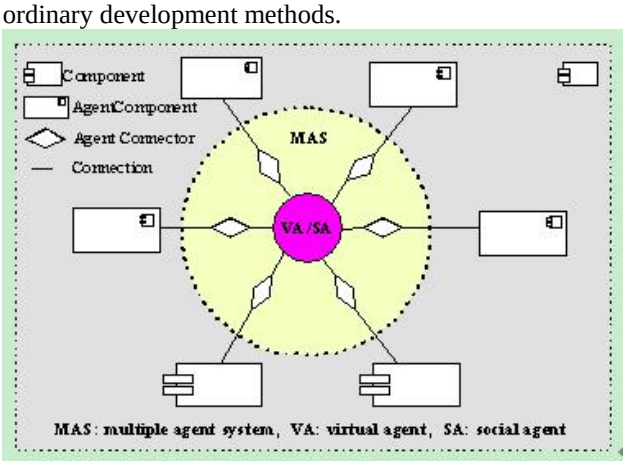

Figure 3. An Agent Component-Oriented MAS General Model

Ⅳ. BASIC CONCEPTS AND CONCEPTUAL PROCESS MODEL

A. Basic Concepts

An object is something that combines the related data with the associated operations on the data. A structured data object means the data information which can be stored in relation database and can be operated by DBMS operations. The form of structured data is one of the most normal data types which are applied in a wide area in the real-world. An agent is a concurrent, autonomous, intelligent and self-contained object. An agent describes its behaviors by itself, which we called self-contained. The goals to achieve and the behaviors to implement are based on the current environment. An agent component is a component as well as an agent. It synthesizes the advantages of the agent technique and component technique and it overcomes the pitfalls of the method of the single agent or component development.

A requirement for an agent-based system can be described in both static requirement and runtime requirement. Static requirement can be extracted from the user requirement specification statically defined. Runtime requirement can be specified by assumption of runtime behavior, and the different behaviors of the system should be assumed and designed in that way.

© 2009 ACADEMY PUBLISHER

Decomposition is to decompose a complex problem into relatively small and manageable components. Decomposition level in structured programming is functions and processes. Decomposition level in object-oriented programming is object. We use this method mainly in requirements analysis phase and system design phase to simplify a complex problem.

B. The Conceptual Process Model

As mentioned above, an agent component-oriented system should be designed in the highly flexible manner. Agents should be independent and autonomous. Agents may be cooperative. However their cooperation should be designed in the highly flexible manner. So we can describe the conceptual process model as Figure 4.

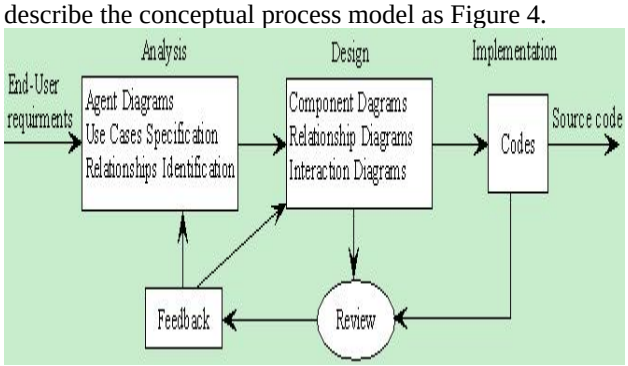

Figure 4. Conceptual process model

- **Analysis:** In the requirement analysis, we gather the customer's requirements from the user requirement specification and use the BDI (Belief-Desire-Intention) model to construct the agent diagram. [19,20].
- **Design:** In the phase of design, we describe the system in details by the models constructed in the phase of analysis. Relationship diagrams show the relationships among agents, such as inheritance, dependency, visibility and logically and physically structured organization. Interaction diagrams show the interactions among several agents, such as local, partial and global agents. Component diagrams show useful information of packaging the related agent components when coding. Component diagrams indicate the implementation group as packages in most programming languages.
- **Implementation:** The next phase is implementation of the models constructed in the previous phases. In the implementation phase, we can select a programming language to code the agent component one by one.

In the previous content, we have simply discussed the method of the agent-oriented software process and then we will discuss the crucial issues for agent component-oriented software process.

V. Structured Data Object-Based Agent Component (SOAC)

A. Agent Component-based Development Technology

Agent component-based development technology (denoted as "ACBD" in the following) is based on building complex software systems by reusable and simpler agent components [21,22]. The ACBD is expected to be capable of reducing development costs and improving the reliability of the entire software system. An agent component is a reusable unit of composition with contractually specified interfaces, which can be used by the outside of the agent component via the interfaces. The ACBD is composed of its two major elements: the agent component architecture and the agent component-based development process. The agent component architecture serves as a standard rule for reusing software agent components. The agent component-based development process is a model of the software development process which emphasizes the central role of the reuse of agent components in the entire development process.

B. Extended the Data Object Control Component

Most of development tools based on components have key and special data object control components (DOCC), such as Data-Grid, TDBGrid [23] etc. A TDBGrid is one of the most important visual data control components provided by Borland Company in a component development tools of Delphi, which can save/restore data from a database conveniently and display the data in the window visually. These kinds of components technologies are provided by so many component-oriented programming languages. They are very useful for developers.

The DOCC has extended features, including the capability of displaying multi-line word wrap column titles, a convenient selection of records from the keyboards, an opportunity to exclude inserting and deleting of records in the Data grid, own standard Popup-Menu, fixing of columns, saving/restoring of a column state, and processing of additional events etc.

In order to make use of the properties and advantages, we should modify or extend the functions of the DOCC easily. Figure 5 illustrates the process of extending. Through the combination with the intelligent agent technology, we can obtain an agent-based data control component, which is visualized, intelligent, common and structured data object-based.

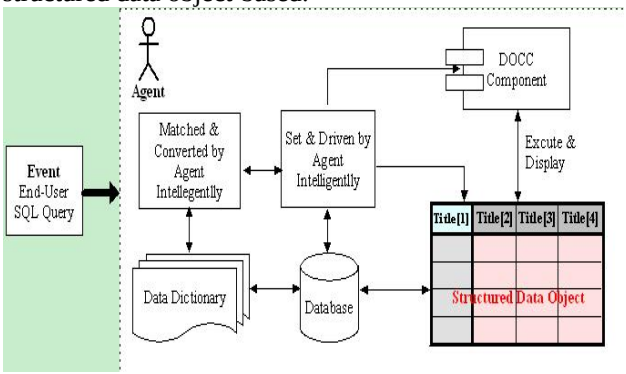

Figure 5. Agent-Based Data Control Component

The extended agent-based data control component has the following features which are different from those of usual DOCC.

© 2009 ACADEMY PUBLISHER

- **Data source choice:** Developers or End-Users only need to choose the data object (a table or a view), which will be displayed in window and denote the related SQL statements.
- **Agent Interface:** An Intelligent Agent component is in charge of checking the SQL statements. Through the index from data dictionary, it converts and sets up the related Title, Caption of each data object column.
- **Several data operation:** It implements some data operations like browse and update of the dataset displayed in the data control component in order to update the database.
- **Data format conversion component:** In order to ease the data management and operation for the users, a data file conversion component is specifically designed, which can convert the data of SQL server into those of DBF or Excel.
- **Print & print setting component:** To print out the data of data object, the QuickReport component has been designed, which can automatically draw lines of a table in a report form, set the height of row manually or automatically, and make the column width automatically set in accordance with the length of data or the width of the data-grid displayed in the window.

C. Defining a SOAC

After examining its basic properties, it is possible to create a single, comprehensive definition of SOAC: A SOAC is a language-neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces. A SOAC is not platform-constrained or application-bound. A SOAC should be autonomous, reactive, pro-active and social competent.

A SOAC is considered to be language-neutral. This does not mean that all agent components are universally written in a single language. It means that they are specially designed and deployed so that agent components written in different languages can work together, which effectively makes them language -neutral.

A SOAC may be implemented independently. This is possible because agent components are encapsulated -- each one has its own self-contained small unit of development and testing.

A SOAC is not constrained to a single platform. It is possible to create different settings for a SOAC so that it can be operated on any platforms.

A SOAC is not bound to any particular application. Although many agent components are created to meet the needs of a particular application, once they have been built and deployed, it is possible to use them for different applications. As long as the interfaces of the agent component meet the consumer's needs, the same agent component can be used to develop or enhance other applications. With some basic features of an agent, a SOAC is an agent as well as a component. A SOAC is based on a structured data object and is agent component-oriented.

D. An Agent Component Architecture Model

A component assembly framework supports a variety of visualization and programming tools for developing component connections [24]. Most of the current systems assume that the user has medium for advanced computers and programming experience. Moreover, the user is required to have the information about the details of each software component [25]. However, the trend is that users will be domain specialists other than necessary computer experts. Further, the End-Users have become more involved in the application development. The component-based software development will let the domain specialists-- End-Users-- use specific domain components and integrate them. Therefore, common software architectures for reusable software components should be developed into such an architecture that can implement the functions of effective design and development in such an environment.

Reuse of components cannot be achieved if there are no standard software architectures. The agent technology is employed in the architecture to satisfy the specified user requirements. The agents are not only interfaces; they are knowledgeable on process and specialized in a certain area. Furthermore, an agent acquires the knowledge to decide when to help the user, know what to do to help the user and how to help. Although they are autonomous entities in the system, they will work together with the user and for the user. In order to satisfy the user's requirements listed above, the following architecture shown in figure 6 is proposed. In the proposed architecture, each agent has three parts:

- **Attributes:** They identify the agent. Among them are the specialization of the agent, owner, success level, life cycle, and implementation environment. This list is not exhaustive.
- **Behavior model:** It specifies how the agent operates and when it terminates. Here, the rules for security, the relationship among the agent, the user and/or other agents is set. These rules also determine how to evaluate the performance of the agents. The agents are autonomous and independent. It is very important to have trustworthy agents. Therefore, the behavior model will ensure that the only agents would be well-behaved and can continue to operate usefully.
- **Inference Engine:** It operates the agent-based components on the behavior model.

According to the content introduced above, we can construct an agent component design and implementation model as it is shown in Figure 6.

- User Agent interface hides the complexities of the system operations. It allows the user to define his/her one-time queries as well as long term, relatively static needs.
- Domain Agent works on behalf of the user. It is responsible for representing user's requirements. It formulates the component specifications from the user requirements and delegates the authority to the domain agent mediator to find the matched software components. It evaluates the performance of domain agent mediator based on how to satisfactorily find the

© 2009 ACADEMY PUBLISHER

software components. Furthermore, it can learn from the executing instance and communicate with other domain agents through domain agent mediator to complete its work for the user.

- Domain Agent Mediator provides communication service to domain agents since they can be implemented and placed in a heterogeneous distributed environment.
- An agent component is a software implementation that can be executed on a physical or logical device. An agent component implements one or more interfaces that are imposed upon it. All components must satisfy certain component contracts. In the architecture, it contains universal component and domain component. The universal component is the basic component and the domain component is a professional component. It usually needs some special knowledge for the domain specialist to develop an application system.
- Component contracts ensure that independently developed components obey certain rules so that components can interact in predictable ways, and can be deployed into standard build-time and run-time environment.
- Coordination service is supported by the architecture so that the domain specialist could master less software developing skills. Furthermore it provides several coordination services such as the transaction service and persistence service.
- Component interface provides the execution methods for the users.
- Component repository has contained all kinds of components and a catalog mechanism should be provided by it.

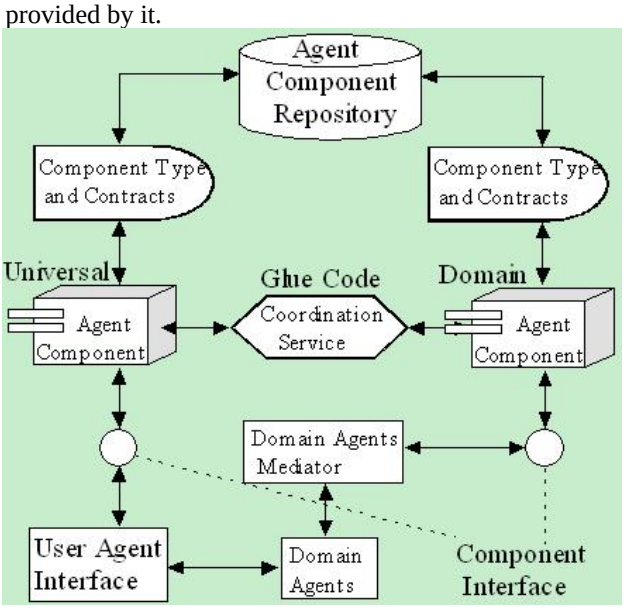

Figure 6. An Agent Component Architecture Model

All of the agents should be self-contained and encapsulated so that security can be achieved to a certain extent although more security measures are needed to avoid the unauthorized uses.

Besides this, a formal specification mechanism should be provided so that a developer can provide some parameters in a formal specification or can do a query operation to a table or a view, he or she would obtain a visualized result or generate a piece of executable code for this application. In the model, the domain component is a special component, which is fit for domain experts to solve some particular problems. It's difficult to solve all the problems arising from different domains in the same way. So, in this kind of component, we only provide a framework for the domain expert, and we hope the domain expert only does some particular jobs with it and writes the results in a database table in the end.

VI. An Application Example

Using the method we have introduced above, we formed a software process model in Figure 4, and we have built an application system in a university information system.

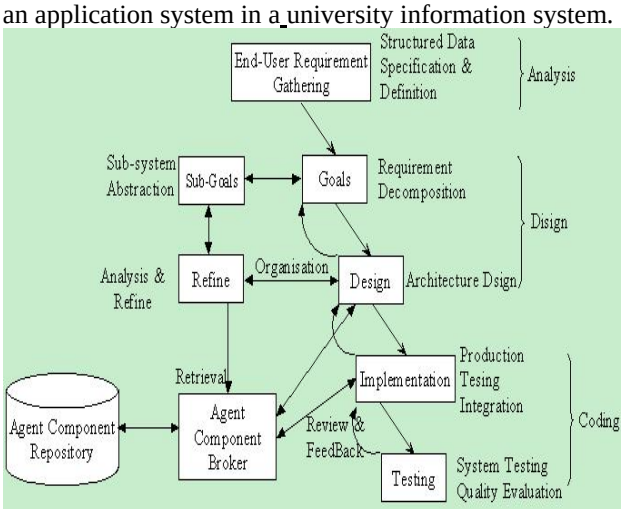

Figure 7. Agent Component-Oriented Software Development Process

According to the methods we have introduced in Sect. 2 and Sect. 3, the architecture of educational management system of ZNU has been established (Fig.8). The system consists of four subsystems, named as management system of status of students, subject achievement management system, elective management system, curriculum schedule management system. Each subsystem is an AC.

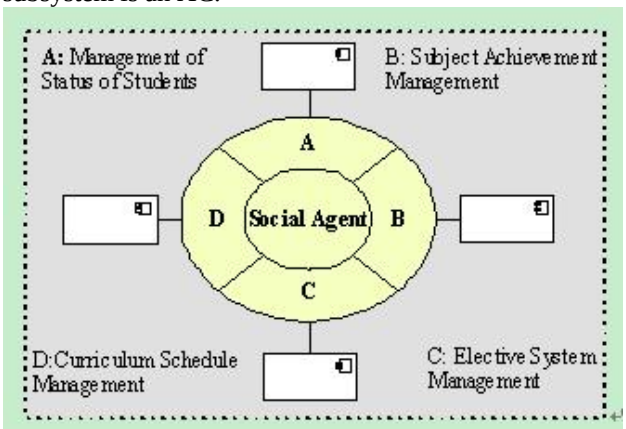

Figure 8. The Architecture of Educational Management System of ZNU

© 2009 ACADEMY PUBLISHER

As an example, to illustrate the detail implementation of AC which we have introduced above, we have developed a subject achievement management system in ZNU. The architecture is illustrated as Fig.9.

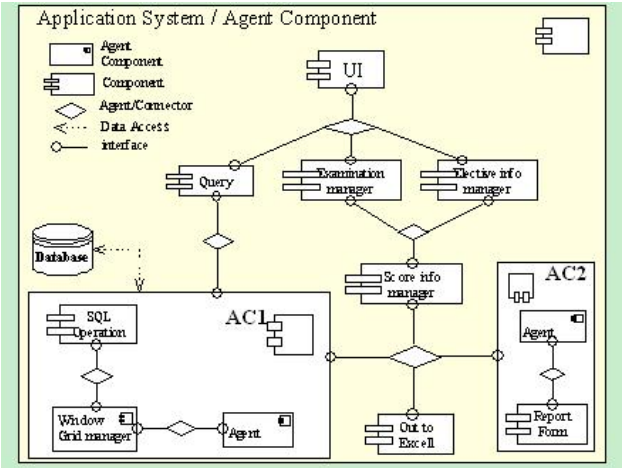

Figure 9. The Architecture of Subject Achievement Management System

In Fig.9, AC1 and AC2 are results refined from the architecture-based analysis and design. AC1 is a data operation agent component, it implements some data operations like browse and update of the data set displayed in the Window Grid component in order to update the database. It consists of three ACs and two agents (connectors); AC2 is a print & print setting AC, it consists of one AC, one report form component and one agent (connector). The detailed description about AC1 and AC2 can be found in [11,17] respectively.

In Fig.9, there are six general components and they are all combined via agent connectors.

Ⅶ. Conclusion

Combining agents and components into an AC component is a new method in the domain of software engineering. In this paper, we propose an AC-based software architecture, and an AC-oriented software development method is introduced. Using the method, we can effectively improve the flexibility, reusability, composability etc. for the complex and large-scale software development.

References

[1] Eunjoo Lee, Byungjeong Lee, Woochang Shin, and Chisu Wu: Toward Component-Based System: Using Static Metrics and Relationships in Object-Oriented System,C.V.

[2] PA Buhler, JM Vidal, Toward the Synthesis of Web Services and Agent Behaviors, AAMAS, 2002, http://jmvidal.cse.sc.edu/papers/buhler02a.pdf

[3] Ladislau B¨ol¨oni, Majid Ali Khan, Xin Bai,Guoqiang Wang, Yongchang Ji, and Dan C. Marinescu: Software Engineering Challenges for Mutable Agent Systems .In: C. Lucena et al. (eds.): SELMAS 2003(2004)149–166.

[4] ML Griss, RR Kessler, Achieving the Promise of Reuse with Agent Components ,SELMAS, 2002, http://martin.griss.com/pubs/selmas2002.pdf

[5] Federico Bergenti,FRAMeTech: A Discussion of Two Major Benefits of Using Agents in Software Development. In: P. Petta et al. (eds.): ESAW 2002(2003)1-12.

[6] Richard Krutisch, Philipp Meier, and Martin Wirsing: The AgentComponent Approach, Combining Agents, and Components. In: M. Schillo et al. (eds.): MATES 2003(2003)1–12.

[7] M. Griss: “Software Agents as Next-Generation Software Components,” Component-Based Software Engineering:Putting the Pieces Together. In: G. Heineman and W. Council(eds.):Addison Wesley Longman, Reading, Mass., (2001).

[8] Hrishikesh J. Goradia and José M. Vidal: Building Blocks for Agent Design. In: P. Giorgini, J.P. Müller, J. Odell (eds.): AOSE 2003(2004)153–166.

[9] Paolo Falcarin and Gustavo Alonso: Software Architecture Evolution through Dynamic AOP. In: F. Oquendo et al. (eds.): EWSA 2004(2004)57–73, 2004.

[10] MEI Hong, CHEN Feng, FENG Yao-Dong, YANG Jie, ABC:An Architecture Based, Component Oriented Approach to Software Development. Journal of Software, Vol.14, No.4(2003)721-732.

[11] Qu Youtian,Chen Tianzhou,Xu Hong: An Agent Component-Oriented Software Process.In: IEEE/WIC/IAT 2005(2005). 459-462

[12] David Garlan:Formal Modeling and Analysis of Software Architecture :Components.In:Connectors, and Events,M. Bernardo and P. Inverardi (eds.): SFM 2003(2003)1–24.

[13] Asuman Sunbul:Abstract State Machines for the Composition of Architectural Styles.In: D. Bj rner, M. Broy, A. Zamulin (eds.): PSI'99(2000) 54-61.

[14] DannyWeyns, Kurt Schelfthout, and Tom Holvoet: Architecture-Centric Development of an AGV Transportation System.In:M. Pechoucek, P. Petta, and L.Z. Varga (eds.): CEEMAS 2005(2005)640–644.

[15] Tomas Bures and Frantisek Plasil: Communication Style Driven Connector Configurations,C.V. Ramamoorthy.In: R.Y. Lee, and K.W. Lee (eds.): SERA 2003(2004)102–116.

[16] Rikard Land: A Brief Survey of Software Architecture,Mälardalen Real-Time Research Center (MRTC) Report:Department of Computer Engineering, Mälardalen University, Västerås, Sweden(2002).

[17] You-Tian Qu, Bing-Yao Jin Hong Xu,Xiao-Tong Ye: A Structured Data Object Based Agent Component Oriented Approach to Software Development,ICMLC2005,270-275.

[18] Griss, M.L. and Pour, G. (2001). Accelerating development with agent components, IEEE Computer, May 2001, 37-43

[19] Jo, Chang-Hyun, A Seamless Approach to the Agent Development, *ACM SAC 2001*, Las Vegas, March, 2001, 641-647.

[20] Chang-Hyun Jo, Jeffery M.Einhom, A Process for BDI Agent-based Software Construction,IMCSE 2003-SERP’03 Las Vegas, Nevada, USA, June 23-26,2003

[21] Larman, Craig, Applying UML and Patterns: Second Edition, *Prentice-Hall*, 2002.

[22] Hironori Washizaki , A Study on Realization of Component-based Software Development Technology , March 2003, Thesis for the Degree of Doctor Graduate Waseda University

[23] The Delphi Companion Component sets, http://www.xs4all.nl/~dgb/comset.html

© 2009 ACADEMY PUBLISHER

[24] Padmal Vitharana,Fatemah “Mariam” Zahedi,Hemant Jain,Component-based Software Development: Design, Retrieval,and Assembly,Revised April 2002

[25] Hironori Washizaki， A Study on Realization of Component-based Software Development Technology， March 2003, Thesis for the Degree of Doctor Graduate Waseda University

Youtian Qu was born in Henghu, Jiangxi Province of China in 1962. In 1986, he received the bachelor's degree in the architecture of Computer science from Nanjing University at Nanjing, Jiangsu province. In 1997 and 2005, as a visiting scholar, he was studying in Zhejiang University and Peking University respectively.

He is a professor and master supervisor at Zhejiang Normal University. His research interests include Component techniques, Agent-Oriented Software engineering, Intelligent Database techniques.

Prof. Qu is the member of IEEE, ACM and senior member of CCF (China Computer Federation).

Chaonan Wang was born in Ningbo, Zhejiang Province of China in 1983. In 2007, she received the bachelor’s degree in computer science and technology from Zhejiang Normal University at Jinhua, Zhejiang province.

She is a postgraduate student of Zhejiang Normal University. Her research interests include trusted computing, Web learning.

Lili Zhong was born in Huzhou, Zhejiang Province of China in 1985. In 2008, she received the bachelor’s degree in computer science from Zhejiang Normal University at Jinhua, Zhejiang province.

She is a postgraduate student of Zhejiang Normal University. Her research interests include trusted computing, Intelligent Database techniques.

Huilai Zou was born in Tianmen, Hubei Province of China in 1986. In 2008, he received the bachelor’s degree in computer science from Huainan Normal University at Huainan, Anhui province.

He is a postgraduate student of Zhejiang Normal University. His research interests include Software Engineering, Artificial Intelligence.

Hua Liu was born in Lishui, Zhejiang Province of China in 1986. In 2008, he received the bachelor’s degree in computer science from Zhejiang Normal University at Jinhua, Zhejiang province.

He is a postgraduate student of Zhejiang Normal University. His research interests include Software Engineering, Artificial Intelligence.

© 2009 ACADEMY PUBLISHER