Development Method for Multi-Agent Real-Time Systems

Lichen Zhang

Faculty of Computer Science and Technology
Guangdong University of Technology , 510090

Zhanglichen1962@163.com

Abstract

Recent developments in multi-agent real-time systems have been promising by achieving autonomous, collaborative behavior between agents in various environments. Agents are being used in an increasingly wide variety of applications from mobile systems to complex mission control and safety systems. However current methods and tools that aid in the development of complex real-time multi-agent systems are still underdeveloped. In this paper, we propose an approach for analyzing and designing multi-agent real-time systems based on UML. The proposed approach is illustrated by an elevator example.

Keyword: Multi-Agent, Real-Time System, Development Method, UML

I. Introduction

Agent systems are among the most complex of systems to develop [1]. The autonomy in the behaviour of the agents contributes inherently to this complexity. The tasks performed by the individual agents can be simple or complex in itself, but the agents' autonomy makes the emergent behaviour of the complete multi-agent system both hard to design and hard to verify. Nevertheless, in many applications it is required that the agents cooperate with each other and the users in a in some sense coordinated manner. Often it is essential to analyze requirements on the behaviour of the overall multi-agent system in relation to behavioural properties of the individual agents, in order to develop a system with the right properties. New applications on the area of distributed real-time computing have added a great deal of complexity to the systems, special requirements on the dynamic adaptation to changing conditions. The development of real-time systems has been following the trend of distribution, mainly because a considerable amount of real-time applications is inherently distributed and dynamic. As example, we can name the network of electronic components in today's automobiles. Applications on this context have requirements on flexibility and self-regulation that are not possibly addressed by the traditional development process explained above. If a failure occurs in one electronic component engaged in an activity that requires co-operation with other components, the outcome of such activity is endangered. Due to their fixed relation and lack of self-regulation, the components could not autonomously modify their run time behaviour in order overcome the problem. One centralized monitoring unit could manage all the elements, but the approach of a decentralized decision architecture has the advantage of no single point of failure and reduced flow of data and commands, due to locally made decisions. The actual development level of micro-controller electronics, concerning processing power, allows the vision of self-regulating electronic components on the context of automotive applications.

The concept of autonomous and self-regulated processes have for long time been addressed in the computer science research area of agents. Agent based software engineering is the most promising conceptional tool available to develop complex systems. In the case of distributed real-time systems, the complexity arises from conflicting issues to be addressed, like predictability and the necessity of flexibility to handle non-deterministic and non-trivial environments, faults and system evolution. This paper claims the principle suitability of the agent approach to real-time distributed systems and presents a vision of how this approach could be realized.

Within multi-agent system development, the emphasis is often on specification of the system architecture that is designed, and on the implementation of this design. If requirements are considered, they are kept implicit or informal. In principle, the required behavioural properties play a heuristic role: the system design is made up in such a manner that the system behaviour does what is needed, although it is not explicitly specified what that means [2] [3].

It seems fair to say those current methods and tools that aid in the development of multi-agent real-time systems are still underdeveloped. Some methods and tools have been proposed for the design of such systems . However, with the current state of practice, none of these methods and tools correctly addresses all kinds of user requirements. Although designers typically commit to one method, they often find that its notation is not rich enough to express some semantic concepts or that it lacks the guidance needed to choose design entities. Consequently, they attempt to borrow useful ideas and notations from other methods [4][5].

The objective of our work is to propose an approach, which supports a single design methodology that covers all phases of the life cycle, ensuring that specific real-time requirements of the software will be met. In this paper, we propose an approach for analyzing and designing multi-agent real-time systems based on UML The proposed approach is illustrated by an elevator example.

II. Development Method for Multi-Agent Real-Time Systems

During the seventies, structured programming was the dominant approach to software development. Along with it, software engineering technologies were developed in order to ease and formalize the system development lifecycle: from planning, through analysis and design, and finally to system construction, transition, and maintenance. In the eighties, object-oriented (OO) languages experienced a rise in popularity, bringing with it new concepts such as data encapsulation, inheritance, messaging, and polymorphism. By the end of the eighties and beginning of the nineties, a jungle of modeling approaches grew to support the OO marketplace. To make sense of and unify these various approaches, an Analysis and Design Task Force was established on 29 June 1995 within the OMG. By November 1997, a standard was adopted by the OMG members called the Unified Modeling Language (UML) [6] [7].

The UML unifies and formalizes the methods of many approaches to the object oriented software lifecycle, including Booch, Rumbaugh (OMT), Jacobson, and Odell . It supports the following kinds of models[8] [9]: **static models-** such as class and package diagrams describe the static semantics of data and messages. Within system development, class diagrams are used in two different ways, for two different purposes. First, they can model a problem domain conceptually. Since they are conceptual in nature, they can be presented to the customers. Second, class diagrams can model the implementation of classes—guiding the developers. At a general level, the term class refers to the encapsulated unit.

The conceptual level models types and their associations; the implementation level models implementation classes. While both can be more generally thought of as classes, their usage as concepts and implementation notions is important both in purpose and semantics. Package diagrams group classes in conceptual packages for presentation and consideration. (Physical aggregations of classes are called components which are in the implementation model. **dynamic models-** including interaction diagrams (i.e., sequence and collaboration diagrams), state charts, and activity diagrams. **use cases-** the specification of actions that a system or class can perform by interacting with outside actors. **implementation models-** such as component models and deployment diagrams describing the component distribution on different platforms.

To satisfy the requirements of modeling real-time systems by agent approach, UML should be extended by introducing a new stereotype called <<agent>>. The <<agent>>is a stereotype for the base class <<class>> that is part of the <<classifier>> element , in order to make sure that <<agent>> has the same behavior as class. Real-time feature is described as an instance of <<agent>> and called TimeAspect.

A. *Timing Model*

The time aspect of real-time systems can be designed independently and expressed as a time aspect model. The time aspect model (shown in Fig.1) can be defined and designed based on the general time model . Also UML extensions should make to express time concepts of the time aspect model as shown in table 1.

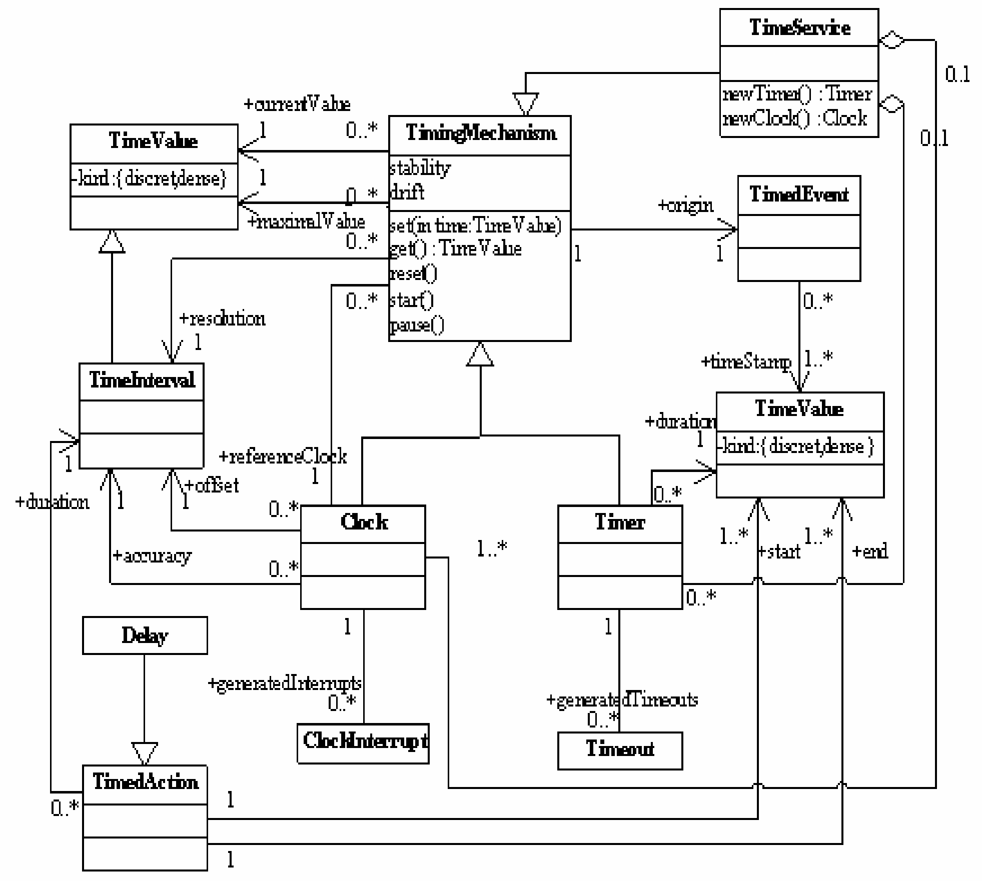

Fig. 1. Time Aspect Model

Table 1. Extension Time Stereotypes

Stereotype	Base Metaclass	Descriptions
<<Ttimer>>	Class	<<Ttimer>> is a kind of timing mechanism that generates one or more timeout signals. The time value indicates the duration that the timeout event goes through from the start to the timeout.
<<Tclock>>	Class	<<Tclock>> is a kind of timing mechanism that generates a clock interrupts periodically. The value of a clock indicates the time that events go through from the start till now.
<<TtimingMechanism>>	Class	<<TtimingMechanism>> is an abstract stereotype that expresses the common features of timer and clock, and performs time measurement and timing-related functions.
<<TtimeService>>	Class	<<TtimeService>> provides the service mechanism for the system
<<TnewClock>>	Operation	<<TnewClock>> creates a clock agent
<<TnewTimer>>	Operation	<<TnewTimer>> creates a timer agent
<<Taction>>	Method	<<Taction>> models any action that takes time using start and end time, or duration

B. *An Elevator System Example*

We consider that every floor has a pair of direction lamps indicating that the elevator is moving up or down. There is only one floor button and one direction lamp in the top floor and the bottom floor. Every floor has a sensor to monitor whether the elevator is arriving the floor. We consider that the elevator is required to satisfy the following timing constraints:

[T1] After the elevator has stopped at a particular floor, the elevator's door will open no sooner than OPEN_MIN_TIME and no later than OPEN_MAX_TIME.
[T2] After the elevator has stopped at a given floor the elevator's door will normally stay open for a STAY_OPEN_NORMAL_TIME. However, if the CloseDoorButton on board of the elevator is pressed before this timeout expires, the door will close but no sooner than STAY_OPEN_MIN_TIME.
[T3] After the door is closed, the movement of the elevator can resume, but no sooner than CLOSE_MIN_TIME, and no later than CLOSE_MAX_TIME.

C. *Structural Description Using Class Diagrams*

The real-time feature of real-time systems can be modeled using UML by extending stereotypes, tagged values, and constraints before. For example, timing constraints can be added on the class to express the time feature in class diagram. But the implementation of the time feature were still scattered throughout, resulting in tangled code that was hard to develop and maintain. So we describe the real-time feature as an independent aspect according to the AOP techniques, and design a time model to realize and manage the time aspect in order to make the system easier to design and develop and guarantee the time constraints. We separate the real-time feature as a TimeAspect, which is an instance of <<agent> in the elevator control system.

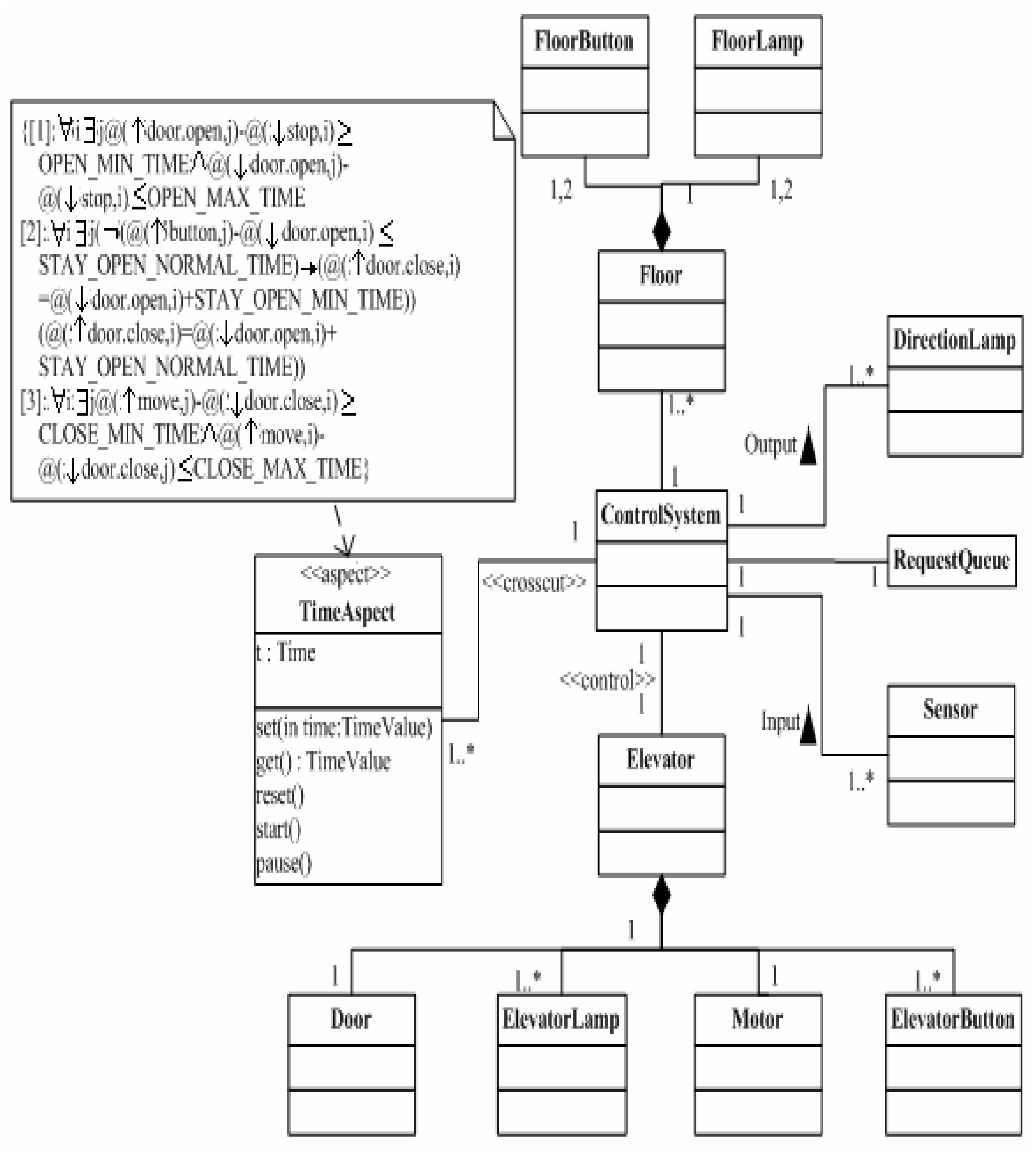

Fig. 2. The Elevator Control System Class Diagram

D. Behavioral Description

UML has five behavioral diagrams to describe the dynamic aspects of a system as representing its changing parts. Use case diagram organizes the behaviors of the system. Sequence diagram focuses on the time ordering of messages. Collaboration diagram emphasizes on the structural organization of agents that send and receive messages. Statechart diagram focuses on the changing state of a system driven by events. Activity diagram focuses on the flow of control from activity to activity messages. Use case diagram, collaboration diagram, and sequence diagram belong to Inter-Agent behavior diagrams. While statechart diagram belong to Intra-Agent behavior diagrams. Collaboration diagram emphasizes on the structural organization of agents that send and receive messages. A collaboration diagram shows a set of agents, links among those agents, and messages sent and received by those agents. It shows classifier roles and the association roles. A classifier role is a set of features required by the collaboration. Classifier roles for core classes implement the core features required by the system. Classifier roles for aspects are services required by the core classes which are otherwise tangled with the roles of the core functional features. The time aspect is time service required by the core classes in real-time systems. The elevator control system expresses the time features as an agent of TimeAspect. The behavior of the time agent interacting on other objects of the system is shown in Fig 3.

Statecharts can model the intra-agent aspectual behaviors well in the agent-oriented programming paradigm. However, current specification of statecharts doesn't support agent-oriented modeling. To support agent-orientation within the context of statecharts, we need to provide a mechanism by which the modeler can express these agents.

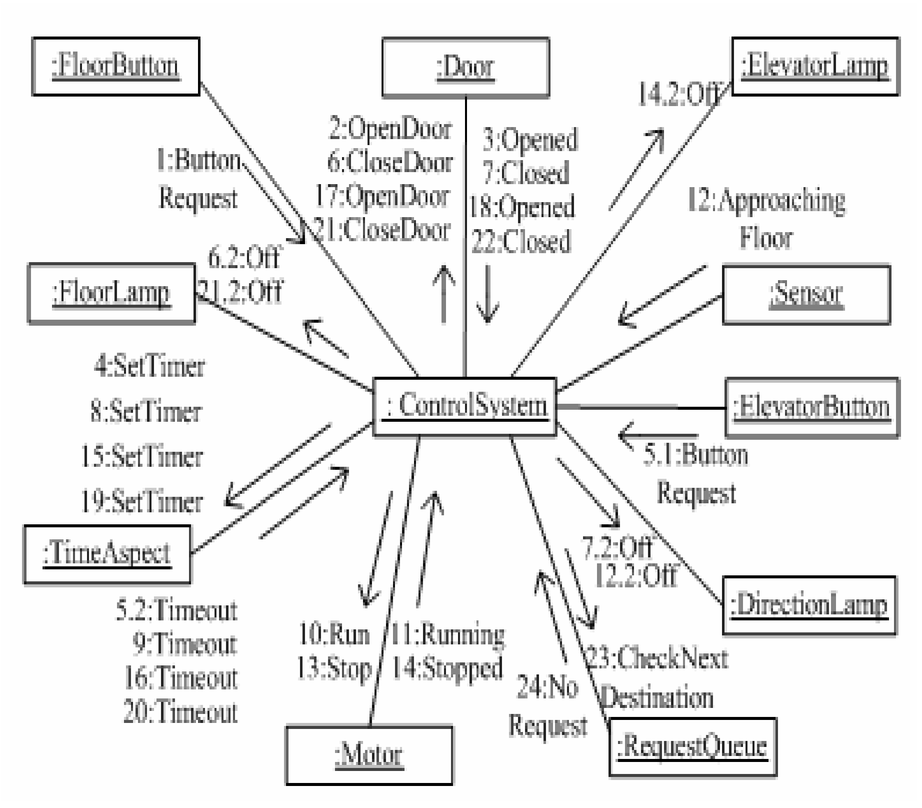

Fig. 3. Collaboration Diagram of The Elevator Control System

Statecharts modeling agents should consider the association between agents and transitions instead of states. Orthogonal regions, which are shown as dashed lines in statecharts, combine multiple simultaneous descriptions of the same agent. Agents have their own sub-states.

Interactions between regions occur typically through shared variables, awareness of state changes in other regions and message passing mechanisms such as broadcasting, and propagating events. The statecharts of the elevator control system is shown in Fig. 4. Timing behaviors are described with the advanced features of Statecharts.

The time aspect can be woven into the real time system by using the UML's statecharts, as statecharts refine the model. Libraries of core and time aspect statecharts can be developed concurrently and independently, and combined only when needed for a particular application.
In this paper, we weave the time aspect with high-level declarations about how an event in the time statechart can be treated like a completely different event in the core statechart. The weaving framework permits statecharts design to be translated into skeleton code for a class. The time aspect and core statecharts objects may be joined to create orthogonal regions. In addition, the time aspect statechart can be woven by specifying which events shall be reinterpreted to have meaning in other Statecharts . The declarations of events reinterpretation of the elevator example can be described as follows according to the timing constraints:

[1] If the core statechart is in the 'ElevatorStopping' state and a 'openDoor' event is introduced, and if the time aspect statechart is in the state 'InitState' satisfying 't>=Min_Time&&t<=Max_Time'. Then the time aspect statechart transfers to the 'Normal' state, the core statechart treats the 'openDoor' event exactly and transfers to the 'DoorOpening' state.

[2] If the core statechart is in the 'DoorOpening' state and a 'closeDoor' event is introduced, and if the time aspect statechart is in the state 'InitState' satisfying 't>=Min_Time'. Then the time aspect statechart transfers to the 'Normal' state, the core statechart treats the 'closeDoor' event exactly and transfers to the 'DoorClosing' state.

[3] If the core statechart is in the 'DoorOpening' state and no any event is introduced, and if the time aspect statechart is in the state 'Normal' satisfying 't>Max_Time'. Then the time aspect statechart transfers to the 'TimeOut' state, the core statechart transfers to the 'DoorClosing' state.

[4] If the core statechart is in the 'ReadyMove' state and a 'down(up)' event is introduced, and if the time aspect statechart is in the state 'InitState' satisfying 't>=Min_Time&&t<=Max_Time'. Then the time aspect statechart transfers to the 'Normal' state, the core statechart treats the 'down(up)' event exactly and transfers to the 'Elevator Starting Down(Elevator Starting Up)' state.

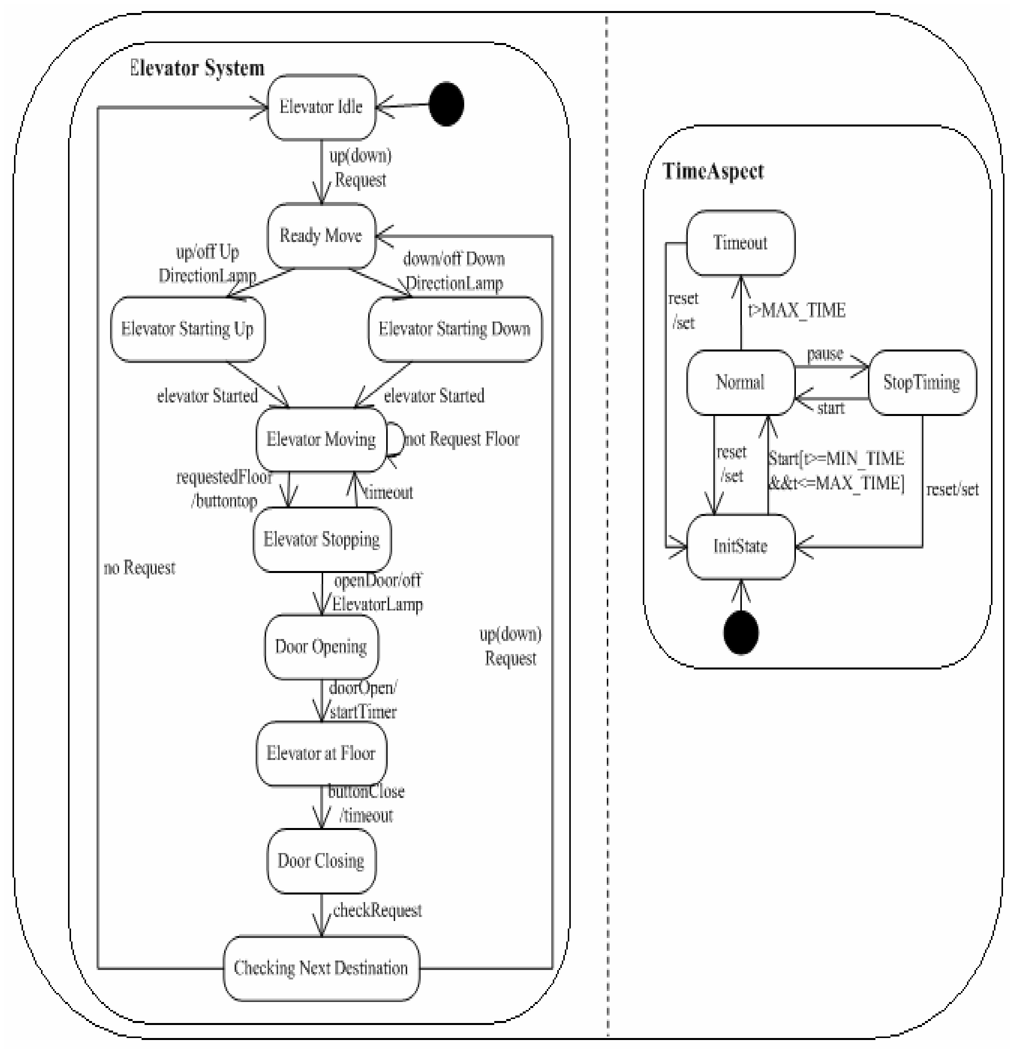

Fig. 4. Statechart of the Elevator Control System

III. Conclusion

We have proposed a development method for multi-agent real-time systems. Based on UML. UML provides tools for :

specifying agent interaction protocols as a whole;
expressing the interaction pattern among agents within a protocol;
representing the internal behavior of an agent,;
representing other agent-related UML extensions that are already commonly used, such as richer role specification, packages with agent interfaces, deployment diagrams indicating mobility, and other techniques.

The real-time feature of real-time systems can be modeled using UML by extending stereotypes, tagged values, and constraints in general. We believe that proposed approach is particularly appropriate for multi-agent real-time systems.

The research reported here is still in progress. Much remains to be done to further refine the proposed approach and validate its usefulness with real case studies. We are currently working on the development of additional formal analysis techniques for our approach including temporal analysis (using model-checking), goal analysis and social structures analysis, also the development of tools that support different phases of the methodology and the definition of the formal specification language.

Acknowledgements

This work is partly supported by the National Natural Science Foundation of China under Grant No.60474072 and No.60174050; the Natural Science Foundation of Guangdong Province of China under Grant No.04009465, the Natural Science Foundation of Guangdong Education Department under Grant No. Z03024.

References

[1] Paulo G. de A. Urbano , Agent Based Approach to Distributed Real-Time Systems Development, citeseer.ist.psu.edu/551418.html.

[2] Krishna Kavi, Mohamed Aborizka and David Kung, A framework for designing, modeling and analyzing agent based software systems, in Proc. of 5th International Conference on Algorithms & Architectures for Parallel Processing, October 23-25, 2002, Beijing, China.

[3] Krishna Kavi, David Kung, Hitesh Bhambhani, Gaurav Pancholi, Marie Kanikarla, Riken Shah, Extending UML to Modeling and Design of Multi-Agent Systems,' Proc. of ICSE'03 Workshop on Software Engineering for Large Multi-Agent Systems (SELMAS'03), Portland, Oregon, May 3--4, 2003.

[4] C. Iglesias, M. Garijo, and J. C. Gonzales. “A survey of agent-oriented methodologies”. In Intelligent Agents V: Proceedings of the ATAL’98, volume 1555 of LNAI. Springer, 1999

[5] Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E. Specification of Behavioural Requirements within Compositional Multi-Agent System Design, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99 (1999)

[6] Bauer, B., Extending UML for the Specification of Interaction Protocols, submitted for the 6th Call for Proposal of FIPA, 1999.

[7] Bauer, B., Extending UML for the Specification of Interaction Protocols, ICMAS 2000, 2000.

[8] Booch, Grady, James Rumbaugh, and Ivar Jacobson, The Unified Language User Guide, Addison-Wesley, Reading, MA, 1999.

[9] Iglesias, Carlos A., Mercedes Garijo, and José C. González, ed., A Survey of Agent-Oriented Methodologies , University Pierre et Marie Curie, Paris, FR, 1998.

[10] Kinny, David, and Michael Georgeff, "Modelling and Design of Multi-Agent Systems," Intelligent Agents III: Proceedings of the Third International workshop on Agent Theories, Architectures, and Languages (ATAL'96), ed., Springer, Heidelberg, 1996.

[11] Kinny, David, Michael Georgeff, and Anand Rao, "A Methodology and Modelling Technique for Systems of BDI Agents," Agents Breaking Away. 7th European Workshop on Modelling

Autonomous Agents in a Multi-Agent World (MAAMAW'96)., Walter VandeVelde and John W. Perram ed., Springer, Berlin, 1996, pp. 56-71.

[12] Singh, Munindar P., ed., Developing Formal Specifications to Coordinate Heterogeneous Autonomous Agents IEEE Computer Society, Paris, FR, 1998.

[13] Wooldridge, Michael, Nicholas R. Jennings, and David Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," International Journal of Autonomous Agents and Multi-Agent Systems, 3:Forthcoming, 2000.

[14] Chung, L. K., Nixon, B. A., Yu, E. and Mylopoulos, J., Non-Functional Requirements in Software Engineering, Kluwer Publishing, 2000.

[15] Jennings, N. R., “On agent-based software engineering”, Artificial lntelligence, 117, 2000, pp. 277-296.

[16] Kinny, D. and Georgeff, M., “Modelling and Design of Multi-Agent System”, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL’96), pp. 1-20, Budapest, Hungary, August 1996.

[17] Odell, J. and Bock, C., Suggested UML Extensions for Agents, OMG document ad/99-1201, Submitted to the OMG’s Analysis and Design Task Force (ADTF) in response to the Request of Information (RFI) entitled “UML 2.0 RFI”, December 1999.

[18] B.Selic, Using UML for modeling complex real-time systems. Lecture Notes in Computer Science, 1474:250-262, 1998.

[19] R.M. Kavi, “Real-time systems: Abstractions, languages, and Design Methodologies”, IEEE Computer Society Press, 1992.

[20] J.A. Stankovic and K.Ramarithm, Hard real-time systems, IEEE computer Society, Order Number819, 1988.

[21] A.M.V. Tilborg and C.M. Koob, Foundations of real-time computing: Formal specifications and methods, Kluwer Academic publishers, 1991.

[22] James Odell, H. Van Dyke Parunak, Bernhard Bauer, Extending UML for Agents,James Odell, Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance, and Eric Yu eds., Austin, TX, pp. 3-17.

[23] Lekshmi S. Nair , Extending ACL to Support Communication in a Real-Time Multi-Agent System , citeseer.ist.psu.edu/482945.html.

Lichen Zhang is a professor at faculty of computer science and technology, Guangdong University of Technology, China. He received his PhD in computer science from Paul Sabatier University, France in 1993. His research interests include distributed systems, computer networks, real-time systems, mobile computing and software engineering. His email address is zhanglichen1962@163.com.