2011 35th IEEE Annual Computer Software and Applications Conference Workshops

Fuzzy Control-based Software Self-Adaptation: A Case Study in Mission Critical Systems

Qiliang Yang[1, 2], Jian Lü[1], Jianchun Xing[2], Xianping Tao[1], Hao Hu[1], Yang Zou[1]
[1.] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
[2.] Engineering Institute of Corps of Engineers, PLA University of Science and Technology, Nanjing 210007, China
yql@smail.nju.edu.cn, lj@nju.edu.cn, xjc@893.com.cn,{txp, myou, zy}@nju.edu.cn

***Abstract*—Self-adaptation ability is particularly desirable for mission critical software (MCS). This paper proposes a fuzzy control-based approach to provide a systematic, engineering, and intuitive way for programmers to achieve software self-adaptation. This approach uses fuzzy logic to handle uncertainty in software, uses natural-language style to describe self-adaptation logic, and makes control visible in software. These greatly shorten the knowledge and semantics gap between control engineering and software engineering, and reduce the realization difficulty of software self-adaptation. We illustrate our approach through the development of an adaptive MCS application in process control systems.**

Keywords-software self-adaptation; fuzzy control; mission critical software

I. INTRODUCTION

The need for adaptability is perhaps most acute in mission critical systems such as process control systems. When facing internal or external changes, *mission critical software* (MCS) shall automatically adjust its states or behaviors in order to provide continuous availability and predefined QoS. Especially, this self-adaptation process of mission critical software must be online and no human oversight. Any interrupted service and downtime of software will lead to very bad consequences such as economic cost and life lost.

Constructing self-adaptive software has been attracted many research efforts in the past several years. Self-adaptive software shifts the implementation style of software applications from open loop to closed loop. It aims to provide an automated and systematic way of handling the increasing complexity and uncertainty of operation management. In fact, *uncertainty* is becoming more and more obvious in software systems. With the increasing complexity of computing systems, we cannot obtain the complete information about the environments or requirements. Several researchers begin to pay attention to uncertainty in software engineering. For example, B.H.C. Cheng et al. advice that traditional requirement descriptions such as "*the system shall do this …*" need to be relaxed and could be replaced with "*the system might do this*" in adaptive requirement engineering [1].

This paper looks at engineering self-adaptive software from a control theory viewpoint. This view has been increasingly discussed in software engineering community from the concept level to the implementation level [2]. The works [3,4] in the concept level propose various models for software self-adaptation based on feed-back control theory. The works [5,6] in the implementation level mostly apply classical control algorithms (most are PID, Proportional-Integral-Derivative) for building self-adaptation strategies. There still exists inadequacy of the current control theory-based research. (1) They neglect the *uncertainty* problem in software and do not provide systematic methods to handle the *uncertainty*. (2) The classical control theory-based approaches usually require the formal model (e.g., differential equation) of software systems, whereas accurately and mathematically modeling software systems is very hard in the uncertain environments. (3)The current approaches lack intuition, and result in a big knowledge and semantics gap between control engineering and software engineering. The design of controllers and control strategies needs too much knowledge of control theory and is very difficult to software engineering researchers and practitioners.

In this paper, we propose a fuzzy control-based approach to achieve software self-adaptation and deal with the uncertainty of software. Our key contribution is threefold. (1) A conceptual framework and a six-step implementation process of fuzzy self-adaptation of software are proposed. (2) We consider the elements in feedback control loop as first class entities in software, and make the control loop visible and explicit, which takes a step to explore the problem of control visibility in adaptive systems proposed by H.A. Müller et al. [7]. (3)We have successfully applied our approach to mission critical process control systems including the development of a self-adaptive OPC Server-FuzzyLon893OPCServer. The rest of the paper is organized as follows: Section II gives the background on a running example. Section III describes our approach for constructing fuzzy software self-adaptation. Section IV illustrates and validates our approach using the running example. Section V discusses related work, and Section VI concludes the paper.

II. A RUNNING EXAMPLE

In previous work, we developed a MCS application called *Lon893OPCServer* [8] for our Lonworks fieldbus network systems. This application is mission-critical and strictly conformed to the *Object Linking and Embedding for Process Control* (OPC) Standard [9], an important specification for the cooperation between real-time process control software. The typical application scenario of the Lon893OPCServer is shown in Figure 1.

The Lon893OPCServer running in the OPC Server station is a key software system to bridge Lonworks fieldbus to supervisory software systems such as HMI software. Each of real-world I/O devices of LonWorks fieldbus is mapped into an I/O device object in the OPC server. The *ScanPeriod* property of every I/O device greatly affects the performance especially the CPU utilization of the OPC Server Station. The small value of the *ScanPeriod* will lead to a high CPU utilization. Traditionally, the *ScanPeriod* property of the Lon893OPCServer is only able to be manually configured before running.

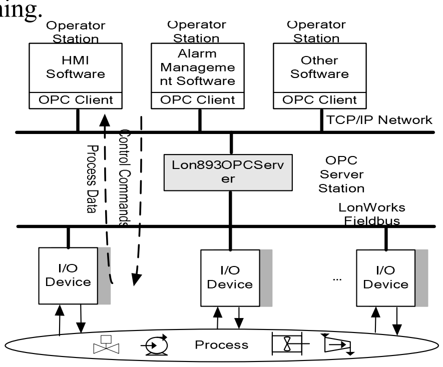

Figure 1. The typical application scenario of Lon893OPCServer

Although our Lon893OPCServer has applied in dozens of projects, it still suffers from a trouble problem: it cannot adapt itself to the uncertain changes of its external run-time environment in OPC server stations. In the application scenario shown as Figure 1, uncertainty faced by the Lon893OPCServer mainly exhibits as the following unexpected changes: the sudden running of the assistant software tools such as LonWorks Manager and anti-virus software in the OPC Server stations, the unpredicted faults of I/O devices, the adding connections of new OPC clients, and so on. All these uncertain changes always consume much CPU utilization and memory of OPC Server stations. The too high CPU utilization of the OPC Server station often causes the bad consequences such as the delay of response time, instability of software system.

Therefore, in order to avoid too high CPU utilization and assure the good performance of the OPC Server station, Lon893OPCServer should have the self-adaptation ability to alleviate the press of CPU through automatically adjusting the parameters or behaviors of it self. As the earlier discussion, we can change the value of CPU utilization by changing the value of the *ScanPeriod* property of I/O devices in the Lon893OPCServer. Due to difficultly obtaining the mathematical relations between the *ScanPeriod* and the CPU utilization, existing model-based methods for self-adaptive software do not work better here. We therefore need systematic ways of achieving software self-adaptation in these uncertain contexts and external environments.

III. Conceptual Framework of our Approach

A. *Overview of the concepetual framework*

We look at software self-adaptation with the concern-separation perspective. In this paper, self-adaptive software is separated into two parts: self-adaptation logic and application logic. The part of self-adaptation logic uses the *sense-decide-act* loop to make software adaptive to external or internal changes. At the same time, the part of application logic presents domain-specific application to meet the function requirements of users. This separation of concerns allows programmers to focus on the development domain-specific business logic in normal ways. Once the self-adaptation ability is required to the software, the self-adaptation logic can be weaved into the target application logic, which has no influence to the old program codes.

This paper, considering the complexity of mission-critical software and the uncertainty of its context, employs the ideas from fuzzy control theory [10] to implement the self-adaptation logic part, and regards the application logic part as the controlled plant. Occasionally, we also name the application logic as target software. Figure 2 illustrates the conceptual framework of our fuzzy-control based approach for software self-adaptation.

As shown in Figure 2, the self-adaptation logic part is a special expert decision system based on fuzzy logic. Sensors encapsulated in software continuously measure the changes from software and its context. These measurements are delivered to a fuzzy controller. The fuzzy controller uses fuzzy logic to make decisions according to the measurements, and takes actions to prevent critical situations. Through actuators, actions are put on the application logic part to regulate the parameters, behaviors, and architecture of it.

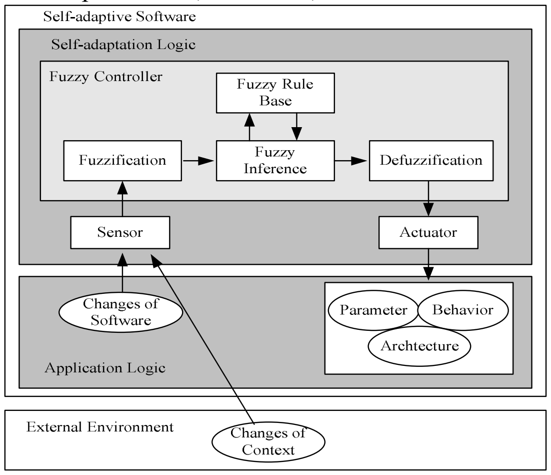

Figure 2. Conceptual framework of fuzzy-control based self-adaptation of software

B. *Three key software entities*

In the framework, sensors, fuzzy controllers, and actuators are the three kinds of key software entities to construct the *sense-decide-act* loop of adaptation.

1) *Sensors*

A sensor in software fuzzy self-adaptation is a software entity to collect data reflecting a specific internal state of the software system or the external runtime environment. The data related to the internal state may be the value of a global variable, a property of class, and other parameters. The data related to the runtime environment may be the performance of operating system and networks (e.g., CPU utilization, throughout, and bandwidth) or even the changes in the natural environment (e.g., air temperature and power voltage). Sensors can be implemented as in vivo or vitro entities using aspect-oriented programming (AOP), object oriented programming (OOP), middleware, and other software technologies.

2) Fuzzy Controller

A fuzzy controller in software fuzzy self-adaptation is a software entity to make decisions on self-adaptation under the guide of the knowledge contained in the fuzzy rule base. The rule-base holds the knowledge of how best to adapt the software system in the form of a set of words-based rules. The knowledge often comes from experienced administrators and designers of the target software systems. The quantitative and crisp data from sensors (e.g., 60% of a sensor for CPU utilization) must be firstly fuzzified into the qualitative fuzzy linguistic values (e.g., *Middle* and *High*) so that they can be interpreted and compared to the fuzzy rules in the fuzzy rule-base.

The inference mechanism evaluates which self-adaptation rules are relevant at the current time and then decides what actions to the target software should be. The decisions made by the fuzzy inference are fuzzy linguistic values. These fuzzy values cannot be directly used to adapt the application logic. The defuzzification is required to convert the fuzzy conclusions drawn by the inference mechanism into the crisp values to actuators.

3) Actuator

An actuator in software fuzzy self-adaptation is a software entity that put actions to other software facilities. Actuators can change parameters (e.g. ScanPeriod in Lon893OPCServer), methods, or the architecture of the target software.

C. Basic steps of achieveing fuzzy self-adadption

In our fuzzy-control-based framework, we propose the corresponding development process for achieving software fuzzy self-adaptation:

(1) Define the goals of self-adaptation. The process of software self-adaptation is always goal-directed. Before the development of self-adaptive software, we should well know what the final goals of the process of software self-adaptation are.

(2) Specify the software facilities related to the goals. Based on analyzing the dynamic behaviors of the target software, identify the software facilities such as software variables, methods, and connectors, which influence the self-adaptation goals.

(3) Construct the fuzzy self-adaptation loop. The fuzzy self-adaptation loop is the feedback control loop in fuzzy control theory, which consists of fuzzy controllers, sensors, and actuators. The specified software facilities in Step (2) are mapped into sensors and actuators.

(4) Convert the control elements of control engineering into software entities of software engineering. The control elements such as fuzzy controllers, sensors, and actuators in the fuzzy self-adaptation loop should be firstly implemented into the software entities such as classes or aspects, and then be embedded to target software. This step makes the control loop visible in software.

(5) Implement schedulers to drive the fuzzy self-adaptation loop. A scheduler is a software engine to drive the fuzzy adaptation loop running in the target software.

(6) Test and optimize. The software with the fuzzy self-adaptation ability must be firstly tested and experimented. During the testing process, the fuzzy self-adaptation rules may be modified and optimized to better achieving self-adaptation goals.

IV. IMPLEMENTATON PROCESS

In this section, we provide the concrete implementation process of software fuzzy self-adaptation, accompanied by the running example of the MCS: Lon893OPCServer. The fuzzy inference system adopted here is Mamdani-type [10].

A. Construct Fuzzy Self-adaption Elememts in Control

Define the goals of self-adaptation. We have known that the old version of Lon893OPCServer cannot adapt itself to the sudden rise of the CPU utilization of the runtime environment. We expect the software can automatically change its some important parameters or behaviors to reduce its CPU utilization when the total CPU utilization of the environment is high because the high CPU utilization often results in the bad performance, e.g. delay of control instructions, in the mission critical systems. Therefore, the fuzzy self-adaptation goal of the Lon893OPCServer is defined as *automatically adjusting its CPU Utilization to maintain a running environment with proper CPU Utilization*.

Specify the software facilities related with the goals. According to Section II, the *ScanPeriod* variable in the Lon893OPCServer highly affects the CPU utilization of it. The small value of *ScanPeriod* will lead to a high CPU utilization. Thus, the software facilities related to the goals are the two parameters: *ScanPeriod*, *Total CPU Utilization* of the running environment. From the control perspective, *ScanPeriod* is the control variable and the *total CPU utilization* (directly called as *CPU utilization* in the follows) is the controlled plant.

Construct the fuzzy self-adaptation loop. Referring to the conceptual framework shown in Figure 2, we construct the fuzzy self-adaptation loop of the Lon893OPCServer. In the loop, there are three control elements: a fuzzy controller, a CPU utilization sensor, and a ScanPeriod actuator. This fuzzy self-adaptation loop is shown in Figure 3.

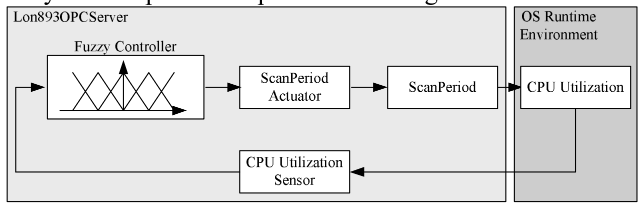

Figure 3. The fuzzy self-adaptation loop of Lon893OPCServer

The CPU utilization sensor is to collect the current value of the CPU utilization of the OS runtime environment. The fuzzy controller is to make self-adaptation decisions in terms of the real-time value from the CPU utilization sensor. The results of the self-adaptation decision are performed by the ScanPeriod actuator to modify the value of the *ScanPeriod* variable in the Lon893OPCServer.

In order to perform fuzzy inference, the input variable *CPU Utilization* and the output variable *ScanPeriod* must be fuzzified. In fuzzy logic, the two variable *CPU Utilization* and *ScanPeriod* are called as linguistic variables.

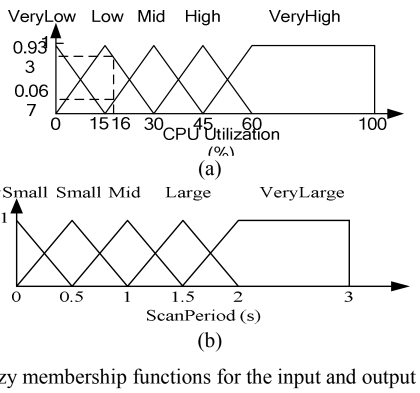

Figure 4. Fuzzy membership functions for the input and output variables: (a) the input variable *CPU utilization* and (b) the output variable *Scanperiod*.

The crisp range of the *CPU utilization* is [0,100] (the unit of it is %). We identify the fuzzy range of the *CPU utilization* as five levels: *Very Low*, *Low*, *Mid*, *High*, and *Very High,* which are called as linguistic values. Membership functions are used to quantify linguistic values in order to automate the fuzzy inference process. The fuzzy membership functions of the above five linguistic values are shown in Figure 4(a). We use triangle functions to describe the four fuzzy values: *Very Low, Low, Mid, and High*, and use a trapezoid function to describe the fuzzy linguistic value *Very High.* Through the fuzzy membership functions, a crisp input value is mapped into one or more fuzzy linguistic values with a degree of certainty. For example, in Figure 4(a), the crisp value of *CPU utilization* 16% is mapped into one fuzzy linguistic value *Low* with the certainty 0.933, and mapped into another linguistic value *Mid* with the certainty 0.067 at the same time. In practice, we find that when the CPU Utilization of the OPC Server station is over 60%, the performance of it often goes bad. Therefore, we specify that when *CPU utilization* ≥60%, the certainty that the value of *CPU utilization* belongs to the fuzzy set ‚*Very High,* is 1.

Considering the practical requirements for Lon893OPCServer, we specify the crisp range of the *ScanPeriod* is [0, 3] (the unit of it is Second). In the same way, we also identify the fuzzy range of the *ScanPeriod* as five levels: *Very Small*, *Small*, *Mid*, *Large*, *Very Large*. The fuzzy membership functions of the five linguistic values are shown in Figure 4(b). The triangle function is used to describe the former four linguistic values, and the trapezoid function is used to describe the fifth linguistic value *Very Large*. We also think that when *ScanPeriod*≥2, the certainty that the value belongs to the fuzzy set ‚*Very Large,* is 1.

Establish the fuzzy rule base. The fuzzy rule base electronically encapsulates the experience from system administrators or designers of target software. It is evaluated using fuzzified measurements in the fuzzy inference phase. The number of rules in the fuzzy rule base lies on the number of the input and output variables. Currently, our fuzzy controller in Lon893OPCServer comprises five rules because there are only one input variable and one output variable. The fuzzy rules for self-adaptation are listed in Table I. These rules are abstracted from our observations on the dynamic characteristics of Lon893OPCServer.

TABLE I. FUZZY SELF-ADAPTATION RULES

ID	CPU Utilization (Premise)	ScanPeriod (Consequent)
R1	VeryLow	VerySmall
R2	Low	Small
R3	Mid	Mid
R4	High	Large
R5	VeryHigh	VeryLarge

Table I consists of 5 *if-then* rules. Column *CPU Utilization* is the premise part, and *ScanPeriod* is the consequent part in a rule. Example rules are as the follows:,

R1: *if CPU Utilization is very low then ScanPeriod is very small*, and

R5: *if CPU Utilization is very high then ScanPeriod is very large*.

The decision process of fuzzy self-adaptation. The decision or inference process is driven by the implications in the fuzzy rules. There are several available inference methods in fuzzy logic. We choose the most popular *max-min method* [9]. Using this method, the fuzzy set specified in the consequent of a rule (e.g., *Large*) is clipped off at a height corresponding to the degree of truth of the rule's premise. After rule matching and evaluating, all fuzzy sets specified in the consequents related to the same output variable are combined with the fuzzy union operation.

The detailed fuzzy self-adaptation process of our Lon893OPCServer is demonstrated as Figure 5. Suppose the value of *CPU utilization* measured from the sensor is 16% at time t_0, then the crisp value 16% is the input of the fuzzy controller.

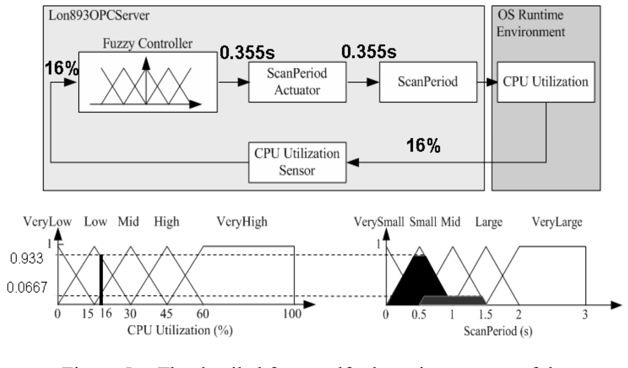

Figure 5. The detailed fuzzy self-adaptation process of the Lon893OPCServer

In the fuzzification phase, in terms of the membership functions for *CPU utilization* defined in Figure 4(a), the crisp value 16% simultaneously corresponds to two fuzzy values: *Low* and *Mid*, and $\mu_{\text{Low}}(16) = 0.933$, $\mu_{\text{Mid}}(16) = 0.0667$ (see the left bottom part in Figure 5). In the fuzzy inference phase, the two fuzzy values, *Low* and *Mid* , activate the two rules, *R2* and *R3*, in the fuzzy rule base listed in Table I. According the max-min method, the rule *R2* recommends the fuzzy value of the output variable *ScanPeriod* is *Small* with the certainty 0.933, and the rule *R3* recommends the fuzzy value of *ScanPeriod* is *Mid* with the certainty 0.0667. The two recommendations are combined using the fuzzy *union* operation, which is shown with the shaded region of the right bottom in Figure 5. In the defuzzification phase, we use the center-of-gravity defuzzification method (COG) [10]. Using the COG, the computed output crisp value is 0.355, and then this value is used to modify the *ScanPeriod* parameter.

B. *Make Control Elements Visible in Software*

In this section, we will discuss the software implementation of the fuzzy self-adaptation. This implementation is the process that mapping the control elements in the control engineering domain into the software entities in the software engineering domain, and also the process of making control elements visible in the Lon893OPCServer. According to the H. Müller' work [7], the visibility of control elements in software means raising a feedback control loop to the architecture level.

Convert the control elements into software entities. We treat the control elements in the fuzzy self-adaptation loop as the first class software entities, and implement these entities using an OO method. This makes the control elements in the software more intuitive and more understandable. The three key control elements in our fuzzy self-adaptation loop are individually mapped into the classes: CSystemInfoSensor, CSystemAcuator, and CFuzzyController.

In CSystemInfoSensor class, a method, *GetCpuState* is defined to collect the current CPU Utilization from the OS runtime environment. Similarly, in the CSystemAcuator class, we define a method called as *ModifyScanTime* to support modifying the *ScanPeriod* property of the global data structure *DeviceRTValue* in our Lon893OPCServer.

The CFuzzyController class provides the three main operations: the fuzzification operation for the input data from the sensor class, the fuzzy inference for the decision, and the defuzzification operation for the output data to the actuator class. We implement the three popular fuzzification methods, *FuzzyTriangle*, *FuzzyTraperzoid*, and *FuzzyGrade* in the fuzzy controller class. The common fuzzy inference operators such as *FuzzyAND*, *FuzzyOR*, *FuzzyNOT* are also implemented in the class. The fuzzy rule base is implemented as a *private* two-dimension array *Fuzzy_Rule_Base* in the controller class. The COG defuzzification method is coded in the *Defuzzification* method of the controller class.

Implement the scheduler to drive the fuzzy self-adaptation loop. In order to drive the fuzzy self-adaptation loop ceaselessly running, a scheduler must be implemented. In our Lon893OPCServer, a thread, *FuzzyControlProc*, is created to serve as the self-adaptation scheduler. The scheduler thread has the same life cycle with Lon893OPCServer: it is born or dead with the starting or exiting of the Lon893OPCServer. In the thread, the instances of the CFuzzyController class, the CSystemInfoSensor class, and the CSystemActuator class cooperate with each other to achieve the self-adaptation goals.

C. *Experiments*

With the aim to validate our fuzzy-control-based approach for software self-adaptation of our Lon893OPCServer, we experimentalize the software with a LonWork fieldbus system. The architecture of the experiment system is like Figure 1. This experiment system consists of one OPC Server station, nine LonWorks I/O modules. As the running platform of the Lon893OPCServer, the OPC Server station is an embedded computer, which is configured with PCM5823 Mainboard, Intel 800 MHz CPU, and 512M memory. The OS of the station is Windows XP Embedded.

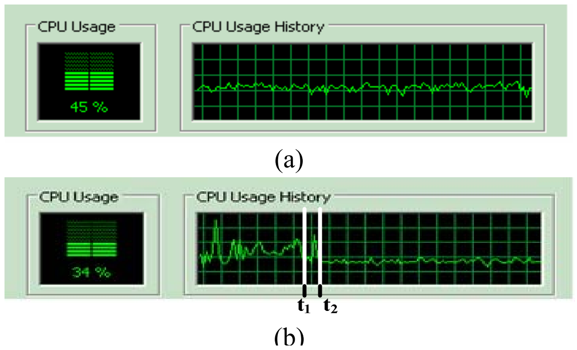

Figure 6. The trend curves of the CPU usage in the experiments of Lon893OPCServer

This experiment intends to compare the CPU utilizations between the old version of Lon893OPCServer and the new one (also named as FuzzyLon893OPCServer) with the fuzzy self-adaptation ability. For ease of comparison, we firstly execute the old version of Lon893OPCServer companied with the intervention of the LMManager software (a debug software tool for the LonWorks network) in the OPC Server station. Figure 6(a) illustrates the varying curve of the CPU Utilization in the OPC Server station for a 5-minute range. In the second step, we stop the running of the old version, and then start the FuzzyLon893OPCServer supporting fuzzy self-adaptation. The varying curve of the CPU utilization in the process is recorded in a 5-minute range too, which is shown in Figure 6(b).

Figure 6 (a) shows the CPU utilization of the OPC Server station keeps a high value about 45% when the Lon893OPCServer with no fuzzy self-adaptation ability runs under the intervention of the LMManager software. In Figure 6(b), under the same intervention of LMManager, at the time t_1, the old version of the Lon893OPCServer has exited, and the FuzzyLon893OPCServer is starting. After a short peak, the FuzzyLon893OPCServer becomes stable since the moment t_2. At the steady state, the CPU utilization is about 34%. Hence, Comparing to the previous version, under the same intervention, our FuzzyLon893OPCServer can effectively reduce the load of CPU by 11%.

V. RELATED WORK AND DISCUSSION

A. *Related work*

The work presented in the paper has been greatly influenced by several related works. On the aspect of control-theory-based software self-adaptation, M. Kokar et al. [3] propose the concept of self-controlling software for software development. This paradigm regards the software system as a plant to be controlled based on modeling the plant and the environment as a dynamic system. J. Shen et al. [4] present three software adaptive models derived from open-loop control, feedback control and adaptive control, and provide an application server framework to support the implementations of these three models. Y. Diao et al. [5] use a PID controller to handling the dramatic performance degradation of IBM database management product. On the aspect of rule-based software self-adaptation, Q. Wang [11] proposes a Rule Model to enhance the self-adaptive ability of software. V.V Phoha et al.[12] use supervisory control theory to model the execution of a software application. In web service domain, D. Gmach, et al. [13] use a fuzzy controller for adaptive enterprise service management to remedy exceptional situations by automatically initiating service migration and replication.

Comparing to the above approaches, our fuzzy control-based approach has the following key differences:(1)the above approaches do not consider the technologies dealing with the uncertainty that is becoming a challenging problem for software systems. (2)The above classical control-theory-based approach needs more professional knowledge control theory (e.g., PID). Bad understandings of classical control theory always lead to bad self-adaptation effects of software. Contrarily, our approach is based on fuzzy logic, which is easy to be grasped and even familiar to a good few of software researchers. Our approach debases the development difficulty of software self-adaptation. (3)The above rule-based approach lacks the formal foundations of logic or mathematics, requires more rules than the one in our work when settling same problems.

This paper extends our previous work[14] with the adaptability requirements of MCS. This paper implements a running adaptive MCS, and employs the MCS as a case study to concretely illustrate and validate our approach.

B. *Discussion*

Our work in the paper is still at the preliminary stage and suffers from several limitations. (1) The fuzzy-control-based framework belongs to the set of internal approaches [15], which needs to intertwine application logic and the adaptation logic, so it has a precondition that we can obtain the source codes of the target software. (2)Our approach only considers the current states of changes, and does not care the history states of software. This may lose knowledge for self-adaptation decisions. (3)The running example in the paper and the rule base are rather simple.

VI. CONCLUSION

Motivated by the mission-critical applications, we have proposed a fuzzy-control-based approach for achieving software self-adaptation. The conceptual framework of software fuzzy self-adaptation and the six-step method for the developments of fuzzy self-adaptation has been introduced. Through a MCS case study, we have described how to use the approach to generate executable software with self-adaptation ability. The results of the experiments show the Lon893OPCServer can online adjust the behaviors of itself to react to the interventions or changes from external runtime environments. This makes the MCS keep good performance and ensures its continuous service ability.

In the future, we will focus on exploring technologies for dynamically injecting software entities including fuzzy controllers, sensors, and actuators, into target software.

ACKNOWLEDGMENT

This work is supported by the National Grand Fundamental Research Program of China (973) under grant 2009CB320702, and the National Natural Science Foundation of China under grants 60736015, 61073031, 60973044 and 61003019.

REFERENCES

[1] B. H.C. Cheng, R. D. Lemos,H Giese,et al., “Software Engineering for Self-Adaptive Systems: A Research Roadmap”. LNCS 5525, 2009, pp.1-26.

[2] K Y Cai, J W Cangussu, R A DeCarlo, A P Mathur, "An overview of software cybernetics", Software Technology and Engineering Practice 2003 Eleventh Annual International Workshop on (2003),pp. 77-86.

[3] M. M. Kokar, K. Baclawski, and Y. A. Eracar, “Control theory-based foundations of self-controlling software”. IEEE Intelligent Systems, Vol.14, No.3, 1999, pp. 37-45.

[4] J. Shen, Q. Wang, and H. Mei, "Self-adaptive Software: Cybernetic Perspective and an Application Server Supported Framework", Proceedings of the 28th Annual International Computer Software and Applications Conference, Sept.28-30, 2004, Hong Kong.

[5] Y. Diao, J. L. Hellerstein, S. Parekh, and et al., "A Control Theory Foundation for Self-Managing Computing Systems", IEEE Journal on Selected Areas in Communications, Vol.23, No.12, 2005, pp.2213-2222.

[6] X. Peng, B. Chen, Y. Yu, W. Zhao, “Self-Tuning of Software Systems through Goal-based Feedback Loop Control”, Proceedings of the 18th Intenaltional Conference on Requirements Engineering, Sept.27-Oct.1 2010, Sydney, pp.104-107.

[7] H.A. Müller, M. Pezzè, and M. Shaw, “Visibility of control in adaptive systems,” Proceedings of the Second International Workshop on Ultra-Large-Scale Software-Intensive Systems, 2008.

[8] J. Xing, Q. Yang, and P. Wang, “Design and Application of Fieldbus OPC DA Server”, Proceedings of the 2010 International Colloquium on Computing, Communication, Control, and Management, August 20-22, 2010, Yangzhou.

[9] J. Liu, W. Ho, K. Tan, et al, “Using the OPC Standard for Real-Time Process Monitoring and Control”, IEEE Software, No.6, 2005, pp.54-59.

[10] L. Wang, A course in Fuzzy Systems and Control, Prentice Hall,1996.

[11] Q. Wang, “Towards a Rule Model for Self-adaptive Software,” SIGSOFT Software Engineering Notes, Vol.30, No.1,2005.

[12] V.V. Phoha, A.U Nadgar, A. Ray,et al,"Supervisory control of software systems",IEEE Transactions on Computers,Vol.53,No.9, 2004.

[13] D. Gmach, S. Krompass, A. Scholz et al., “Adaptive Quality of Service Management for Enterprise Services,” ACM Transactions on the Web, Vol. 2, No. 1, Feb. 2008, pp.8:1-46.

[14] Q. Yang, J. Lü, J. Li, X. Ma, W. Song, Y. Zou, “Towards a Fuzzy-Control-based Approach to Design of Self-adaptive Software”, Proceedings of 2010 the Second Asia-Pacific Symposium on Internetware, Nov. 3-4 2010, Suzhou.

[15] M. Salehie and L. Tahvildari. “Self-adaptive software: landscape and research challenges”, ACM Transactions on Autonomous and Adaptive Systems, Vol.4, No.2,2009,pp.14:1-42.