Toward a Fuzzy Control-based Approach to Design of Self-adaptive Software

Qiliang Yang$^{1, 2}$, Jian Lü^{1}, Juelong Li^{2},
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

Xiaoxing Ma^{1}, Wei Song^{1,3}, Yang Zou$^{1, 4}$
3. Institute of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing210094, China
4. Computer and Information College, Hohai University, Nanjing 210098, China
xxm@nju.edu.cn

ABSTRACT

Self-adaptive software is expected to adjust itself attributes or structures at runtime in response to changes. Aiming at addressing some challenging problems such as difficult mathematically modeling software using the current control theoretical methods, we propose a novel fuzzy-control-based approach to achieve self-adaptive software, which is presented as framework of fuzzy self-adaptive software (FFSAS). In this framework, the general model, the implementation architecture, and the design methodology are put forward and discussed in detail. The fuzzy-control-based approach is evaluated with a news-website case study.

Categories and Subject Descriptors

D.2[Software Engineering];D.2.10[Design]:Methodologies;

General Terms

Software engineering, software methodology.

Keywords

Self-adaptive Software, Fuzzy Control, Control Theory.

1. INTRODUCTION

Software is facing the increasing complexities resulted from itself and its context. By the self, the architecture of it is more and more heterogeneous and complex (e.g., Ultra Large Systems and Hybrid Systems). By the context, the external running environments and user needs of software are more uncertain, more dynamic and change more frequently (e.g., Pervasive Computing Systems and Mobile Computing Systems). These twofold complexities lead to the emerging of a new more flexible software model: self-adaptive software. Self-adaptive software aims to add the self-adaptation ability to software, which makes software be able to automatically adjust its properties and behaviors to adapt in the face of changing internal structures, user needs and environments.

Because of sharing the same feedback ideas with self-adaptive software, control theory has been employed by a few of researchers as an important paradigm to address the issues of self-adaptive software. Mary Shaw [1] presents a software design paradigm based on process control. This control paradigm changes the open-looped traditional software model into the closed-looped model that can prevent external disturbances. Kokar et al. [2] give a self-controlling software model based on control theory for specifying and designing software that control itself as it operates. J. Shen et al. [3] present three software adaptive models derived from open-loop control, feedback control and adaptive control.

There are several limitations of the current research based on control theory. Firstly, it requires the model of software, whereas accurately modeling software systems is nearly impossible. Secondly, the design of controllers and control strategies needs too much knowledge of control theory and it is very hard to computer science researchers and practitioners.

In this paper, aiming to address some problems of the current research, we propose a novel fuzzy control-based approach to realize self-adaptive software. Fuzzy control [4] utilizes heuristic information to construct control strategies and controllers, which is tolerant of imprecise data, very conceptually easy to understand. The fuzzy control process is very similar to the reasoning process of human. Therefore, fuzzy control is very feasible for the study on self-adaptive software with the characteristics of vagueness and uncertainty. Our fuzzy control-based method for building self-adaptive software does not require models of software systems, and can make the implementation process more understandable and more convenient. In addition, this method can be supported with a full set of development and simulation tools such as MATLAB in the control community. We name the self-adaptive software based on fuzzy control as *fuzzy self-adaptive software* (FSAS).

Generally, our contributions of this paper can be described as follows: (i) we introduce fuzzy control theory to the research of self-adaptive software, and propose the research concept of *fuzzy self-adaptive software*. (ii)The framework of *fuzzy self-adaptive software* including the general model, the implementation framework, and the design methodology is systematically provided and discussed. (iii)We conduct extensive evaluation to validate our fuzzy-control-based approach by a case study.

2. FRAMEWORK OF FUZZY SELF-ADAPTIVE SOFTWARE

The principles and techniques invented in fuzzy control offer a natural architecture for building self-adaptive software systems. In this section, we give our framework of fuzzy self-adaptive software (FFSAS). The FFSAS is made up of three parts: the general model, the implementation architecture, and the design methodology.

2.1. General model of FFSAS

Based on fuzzy control, we propose a general model for achieving self-adaptive software. This model, as depicted in Figure 1, consists of two layers: the adaptation logic layer and adaptable system layer. The two layers communicate with each other via the software bus. In this model, the adaptation logic is independent from the software application, so our FFSAS belongs to external adaptation approaches.

Adaptation logic layer. This layer provides the adaptation engine and logic to drive the target software system to evolve. The adaptation strategies are realized with fuzzy rules and stored in the fuzzy rule-base. The adaptation engine employs the fuzzy inference mechanism to make adaptation decisions. There are two categories of inputs in the adaptation logic layer: adaptation objectives (i.e., reference inputs) and plant inputs. Adaptation objectives are the goals to which the target software systems will be evolved. Adaptation objectives are often constants. For instance, while user-click amounts changing, in order that the CPU load of every server in a cluster is no more than 60%, we should dynamically adjust the quantity of servers in the cluster. The number 60% in the example is namely the adaptation objective. Plants inputs come from the sensors instrumented in the target software systems. Plants inputs may be current values of the critical variables in the target software systems. In general, the errors between the adaptation objectives and plant inputs are fuzzified as the preconditions of fuzzy inference.

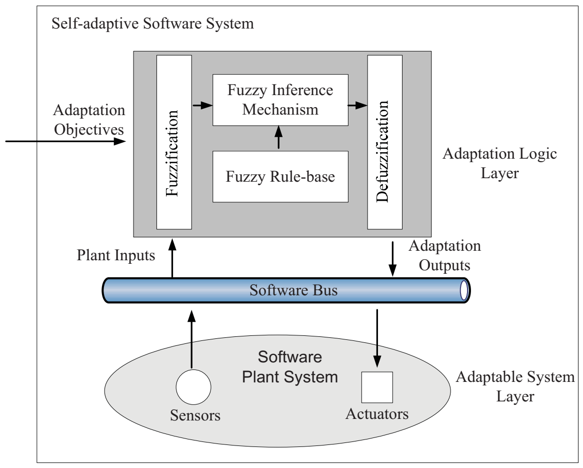


Figure 1. Model of fuzzy self-adaptive software

The adaptation outputs of the adaptation logic layer are the defuzzfied conclusions of fuzzy inference. The outputs put effects on the target software system via actuators.

In a word, the whole adaptation logic layer serves as a fuzzy controller to manipulate the critical properties of software systems in the adaptable system layer.

Adaptable system layer. This layer provides application functions of general software systems. Moreover, sensors and actuators are instrumented in the software system for adaptation. Therefore, the software system is adaptable. From a control-theoretical perspective, the software system is equal to the plant in a control system. Therefore, we also name the adaptable software system as the software plant system.

Software bus. The bus is a logic and conceptual communication facility. It provides unified communication protocols for the interoperations between the adaptation logic layer and the adaptable system layer. With the help of the software bus, the two layers can be separately run in different processes and even different host machines.

Our fuzzy-control-based model adopts a typical non-reflection structure to adapt software systems. In other words, this model does not require to understand and model the software systems. It just needs to know the external dynamic characteristics of the software systems and then describe these characteristics using fuzzy logic. Therefore, this FFSAS model can simplify the course of achieving self-adaptation. Moreover, we can also supervise the whole adaptation process through SCADA (Supervisory Control and Data Acquisition) tools such as iFix, Intouch in the industrial control community, which makes the adaptation process transparent.

2.2. Implementation Architecture of FFSAS

Architecture is an instance of model. Under the guide of the FFSAS model, we propose an implementation architecture for self-adaptive software using popular tools in the control community. The framework is shown in Figure 2.

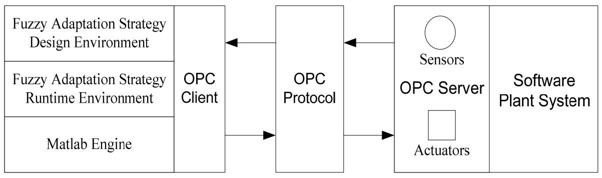

Fuzzy Adaptation Strategy Design Environment
Fuzzy Adaptation Strategy Runtime Environment
Matlab Engine
OPC Client
OPC Protocol
Sensors
OPC Server
Actuators
Software Plant System

Figure 2. The design framework of FFSAS

Generally, corresponding to the model of FFSAS, as depicted in Figure 3, we implement the adaptation logic layer with MATLAB, implement the software bus with OPC (OLE for Process Control) standards [6], and encapsulate sensors and actuators as OPC servers in the adaptable system layer. The Fuzzy adaptation strategy can be easily designed with FIS (Fuzzy Inference System) Editor, interpreted and executed by stand-alone Fuzzy Engine, and simulated with Simulink in MATLAB.

The Software bus is realized with OPC, which is the very popular software interoperation protocol in the control community. OPC defines a series of standard interfaces. Using these interfaces, clients can communicate to various software products under the universal data access standards.

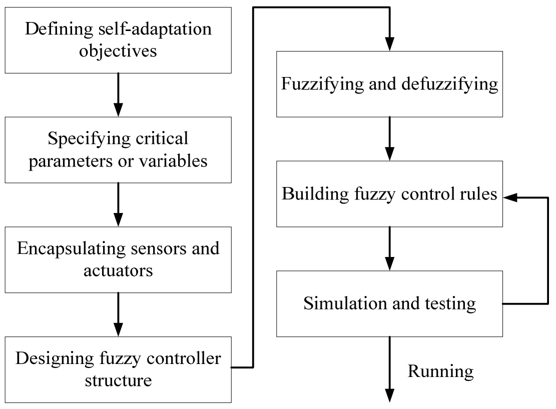
We use the OPC standards to build sensors and actuators, and name these sensors and actuators as OPC sensors and OPC actuators. OPC sensors and OPC actuators are all called OPC servers as well. The architectures of these kinds of sensors or actuators include two parts: OPC wrappers and primitive sensors (actuators). OPC wrappers provide the standard OPC access interfaces for other software, and perform the data format translation between the primitive sensors (actuators) and OPC interfaces. Primitive sensors (actuators) are software entities that can directly monitor and collect information (or effect the states and behaviors) of software plant systems. Encapsulating sensors and actuators with OPC standards in software systems provides many advantages to our FFSAS model: many sophisticated software tools supporting OPC in industrial control area such as SCADA software, etc. can be employed to monitor, control, and analyze self-adaptive software systems.

No particular primitive sensor or actuator technology is provided for the design framework of the FFSAS in the paper. We can employ a number of different sensor or actuator solutions developed by others such as injecting callbacks into source codes [7], modifying byte codes or binaries, replacing DLLs or other dynamic libraries, monitor operating system resource usage, and so on.

2.3. The design methodology of FFSAS

Combining with the features of both software engineering and fuzzy control, we propose a design methodology to achieve fuzzy adaptation of software systems. The basic idea of this methodology is to treat software systems as control plants, and then maps the software self-adaptation problem into the fuzzy control problem.

The design process of software fuzzy adaptation can be divided into the following seven key steps: ①defining self-adaptation objectives, ②specifying critical parameters or variables in the software plants, ③encapsulating sensors and actuators ,④designing fuzzy controller structure, ⑤fuzzifying and defuzzifying, ⑥building fuzzy control rules, ⑦simulation and testing. Figure 3 illustrates the implementation flow of software fuzzy adaptation. Corresponding to the model of FFSAS, Step1 to Step 3 concerns the design of the adaptable system layer, and Step 4 to Step 7 concerns with the design of the adaptation logic layer. In Step 7, if the results of self-adaptation cannot meet the needs of users, Step 7 must repeatedly jump to Step 6 for redefining and modifying the fuzzy control rules until the results of simulation and testing are good.



Defining self-adaptation objectives
Specifying critical parameters or variables
Encapsulating sensors and actuators
Designing fuzzy controller structure
Fuzzifying and defuzzifying
Building fuzzy control rules
Simulation and testing
Running

Figure 3. The design flow of software fuzzy adaptation

Step 1: defining the self-adaptation objectives. When adding self-adaptation ability to software plants, we must firstly analyze the self-adaptation requirements of the software systems based on the contexts of the systems and define the objectives to which the software systems should move. The self-adaptation objectives are also referred as reference inputs or set points. They are the desired values of the measured outputs of the software plants, such as *CPU utilization, should be 50%.*

Step 2: specifying critical parameters or variables. We often need monitor or change some parameters (variables) in order to maintain the self-adaptation objectives in the software plants. For instance, in a database server, when realizing the objective that the CPU utilization is less than 50%, we often dynamically change the *max connection number* of the server according to the real-timely measured value of *CPU utilization*. At this time*, max connection number* and *CPU utilization* are the critical parameters or variables. These variables have very close relations with the linguistic variables in fuzzy control.

Step 3: encapsulating sensors and actuators. This step puts emphasis on how to get or change the values of the variables in the software plants. We can employ some mature technologies to implement the monitored parameters as sensors, and the adjusted parameters as actuators. For example, the OS API invoking techniques can be employed to implement the sensor for getting the value of *CPU utilization,* and the file access techniques are utilized to realize the actuator for modifying the value of *max connection number.* In order to easily interoperate with the self-adaptation logic executing platform (e.g. MATLAB), the sensors and actuators should be wrapped with OPC interfaces on the basis of the OPC specifications.

Step 4: designing the structure of the adaptation logic layer. The problem of how to design an adaptation-logic-layer structure in the FFSAS model can be easily mapped into the problem of designing a fuzzy controller. The design of fuzzy controller structure means properly choosing the input variables and output variables (i.e., control variables) of fuzzy controllers based on the critical parameters or variables specified in Step 2. The choice of the input and output variables has direct influence on the linguistic structure of a fuzzy controller. In most cases, the core problems of control are how to eliminate system output errors. Output errors reflect the deviation degree from the measured output to the desired self-adaptation objectives in software systems. Therefore, the input variables in a fuzzy logic controller are typically the output error, output error derivative, output error integral of a software systems, etc.

Theoretically, the larger dimension number of inputs will produce more accurate control effects. However, the too large dimension number also leads to too complicated fuzzy control rules, which make it very difficult to implement the control algorithms.

Step 5: fuzzifying and defuzzifying. This step is mainly to specify the methods to fuzzify and defuzzify. In fuzzy control application, the observed data such as *E* (the error between the self-adaption objective and the plant input) are usually crisp, so the basic function of fuzzification is converting the input data into suitable linguistic values that may be viewed as labels of fuzzy sets. Experience with the design of a fuzzy logic controller suggests the following two principal ways of dealing with fuzzification.

(1) Discretization of crisp data. The method partitions the continuous crisp data in a limited range (e.g., [-6, 6]) into server levels, and each level is corresponding to a fuzzy set.

(2) Fuzzification of single value. The method converts a crisp value into a fuzzy singleton within a certain universe of discourse. A fuzzy singleton is a preciese value.

Defuzzifying is a mapping from a space of fuzzy control actions into a space of crisp control actions. A defuzzification strategy aims at producing a crisp control action that best represents the possibility distribution of a fuzzy control action. Many defuzzification methods have been produced such as the max criterion, the mean of maximum (MM), center-of-gravity (COG), center-of-sums (COS), and so on. In our experience, the center-of-gravity method is more popular. The universal computing formula of COG is

$$u^{crisp} = \frac{\sum_{i=1}^{n} b_i \int \mu_i}{\sum_{i=1}^{n} \int \mu_i},$$

where u^{cirsp} denotes the defuzzified crisp output of p, the b_i denotes the center of the membership function (i.e., where it reaches its peak) of the consequent of rule (i), μ_i defines the *implied fuzzy set* for rule (i) (i.e., it is the conclusion implied by rule (i)), $\int \mu_i$ denotes the area under the membership function μ_i,, and n denotes the number of rule in the rule base.

Step 6: building fuzzy rule base. A fuzzy rule base is characterized by a set of linguistic statements based on expert knowledge. The expert knowledge is usually in the form of *if-then* rules. Namely, the fuzzy rule base is a set of *if-then* fuzzy control rules. In an *if-then* control rule, the *antecedent* is comprised of the input variables (e.g., the error or the change of error) of a fuzzy controller, and the *consequent* is comprised of the control variables. Before constructing a rule base, we shall firstly discretize or normalize the universes of discourse of the input variables and control variables, and then do fuzzy partition of the spaces of the two class variables.

As was stated earlier, fuzzy rule base is equipped with experience and engineering knowledge. Consequently, for constructing rule bases, it is very important to know how to get this experience and knowledge. Two approaches are proposed to obtain the software adaptation knowledge of software systems.

The first approach is to interview with the designers of the software system to which the adaptation ability will be added. The designers clearly know the internal mechanisms and implementation principles of the software system. We can use a carefully organized questionnaire to get a fuzzy model of the software process.

The second approach is to ask the skilled administrators or operators of the software systems. Through the administrators (operators) could not know the input-output relations with sufficient precision, they can manage or control such a system quite successfully without having any quantitative models in mind. In effect, a human administrator employs-consciously or subconsciously-a set of *if-then* rules to manage and control the software process.

Step 7: Simulation and testing. The aim of this step is to check the self-adaptation logic using related simulation and test technologies. MATLAB may be the best platform for simulation and testing. During the step, the fuzzy rule base should be redefined or updated if the self-adaptation objectives cannot be achieved.

3. EVALUATION

We employ an example *Znn.com* in Cheng's doctoral dissertation [5] as a case study to demonstrate and validate our fuzzy-control-based approach to achieving self-adaptation. In order to make the evaluation process clearer and more rounded, we experimentalize it from two levels: self-adaptation for single-objective, and self-adaptation for multiple objectives. The results of the evaluation show that our approach can realize the self-adaptation of software with good understandability and easiness. Because of space limit, we do not provide the detailed evaluation process in the paper.

4. CONCLUSION

With the aim to address several challenges in designing self-adaptive software, we bring forward the concept of *fuzzy self-adaptive software*, which is a kind of self-adaptive software system driven by fuzzy control strategy. The FFSAS has been presented, which is made up of the general model, the implementation architecture, and the design methodology. The evaluation is conducted for our approach using a case study, which shows that our FFSAS is advantageous to address the problems of self-adaptation for software systems.

However, we clearly realize that our research on the FFSAS is just in concept and still at preliminary stage. To achieve self-adaptive software, several important aspects such as the sensor-monitor facilities and enabling software tools for the FFSAS, need to be further explored.

ACKNOWLEDGEMENTS

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 and 60721002.

REFERENCES

[1] Mary Shaw, "Beyond Objects: A Software Design Paradigm Based on Process Control." *ACM Software Engineering Notes*, Vol. 20, No.1, 1995, pp.27-38.

[2] 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.

[3] 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.

[4] K. M. Passino and S. Yurkovich, *Fuzzy Control,* Addison-Wesley Longman, Inc., 1997.

[5] S.W. Cheng, Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation. Ph.D. Dissertation, Carnegie Mellon University, USA, 2008.

[6] OPC Foundation. Data Access Custom Interface Standard Version 2.04 [Online]. September 5, 2000, http://www.OPCFoundation.org.

[7] G. Heineman, P. Calnan, B. Kurtz, and et. al. Active interface development environment (AIDE). http://www.cs.wpi.edu/heineman/dasada/.