Autonomic Computing Framework for Cyber-Physical Systems

Ilsun Hong¹, Hyunsang Youn¹, Ingeol Chun², and Eunseok Lee¹

¹School of Information and Communication Engineering
Sungkyunkwan University
440-746 Suwon, South Korea
{ishong, wizehack, leees}@skku.edu

²Embedded S/W Research Division
Electronics and Telecommunications Research Institute (ETRI)
440-746 Daejeon, South Korea
igchun@etri.re.kr

Abstract—The more systems are becoming to be controlled by computing devices. Cyber-Physical Systems (CPS) are expected to be an alternative to the traditional control systems. However, there are several problems to adopt CPS since they cannot adapt to changes in environment and system failures. Autonomic computing concept is widely applied to enable systems to adapt changes at runtime. In this paper we propose an autonomic computing framework for CPS including overall architecture, development process and operation process at runtime. The proposed approach allows developers to implement autonomic CPS more systematically. To validate our approach we implemented the autonomic computing engine and confirmed feasibility of the proposed approach through Home Surveillance Robot scenario.

Index Terms—cyber-physical systems, autonomic computing, autonomic computing framework, self-adaptation, self-healing

I. INTRODUCTION

The more social infrastructure tends to be controlled by computing devices. Cyber-Physical Systems (CPS) are expected to be an alternative to the traditional control systems [2]. However, there are several limitations such as the lack of the abilities to change the system behavior against unexpected changes in the external environment and to adapt for the internal system failures [1]. Autonomic computing, introduced by IBM in 2001, is a computing paradigm, which proposes the system that handles unpredicted problems at runtime without human intervention [5]. Autonomic computing has been widely applied to the systems such as web servers and a large-scale data center since it supports the ability that enables the system to self-configure, self-optimize [7]. These features of autonomic computing can be adapted to complement aforementioned limitation of CPS effectively.

In this paper, we propose an autonomic computing framework for CPS including overall architecture, development process and operation process at runtime. We have improved our previous work [9], which is based on predefined static rule, by using dynamic analysis model and diagnosis model. This enables the system to adapt to changes in environment and system failures with less overhead. We implemented an autonomic computing engine to validate our approach and confirmed feasibility of the framework through Home Surveillance Robot scenario.

II. RELATED WORK

A. Cyber-Physical Systems

Cyber-Physical Systems (CPS) are defined as integrations of computation with physical processes [1]. Physical world is monitored and controlled by embedded computing devices and networks with feedback loops, where physical world affects the control system as well. CPS is expected to be applied to complex control systems such as transportation systems, defense systems, robotic systems, smart grid and facility control system such as nuclear power plant [2]. Since the failure of these systems can result in significant economic losses, physical damage and threats to human life, they must operate dependably, safely, securely and reliably. However, several problems can make CPS unreliable. For example, uncertainty in the environment, security attacks, errors in physical devices and wireless communication can affect overall system robustness, security and safety [3]. Therefore, new paradigm for developing and operating CPS is needed [1-4].

B. Autonomic Computing

Autonomic computing is defined as computing systems that can manage themselves given high-level objectives from administrators [5,6]. This means that autonomic computing system needs a minimum of human intervention [5]. Kephart and Garlan suggested a utility-based autonomic computing framework that calculates system utility with performance and cost [6-8]. Such approaches are applied to a large-scale data center and web server respectively. However, these methods use the limited metrics such as response time and throughput to evaluate the system, which is not applicable to the system with multiple objectives.

III. AUTONOMIC COMPUTING FRAMEWORK

In this section, we will explain our framework including overall architecture, development process and operation process at runtime.
A. Overall Architecture

Autonomic computing framework, illustrated in Fig. 1, contains an autonomic computing engine and CPS elements. One or more CPS elements, which make up the entire CPS, are managed by single autonomic computing engine at runtime. Autonomic computing engine consists of six modules, a knowledge base and a communication handler. Each module references the knowledge base to obtain several models and monitoring modules communicates with sensor and effector in the CPS elements through the communication handler. The roles of each module are as follows:

1) Monitoring Module
Monitoring module is a communication channel between an autonomic computing engine and CPS elements. A sensor in CPS element reports the state of the managed system to the monitoring module. Raw data transmitted from a sensor is interpreted based on monitoring models in the knowledge base and passed to analysis module.

2) Analysis Module
Analysis module determines whether the managed system should adapt after analyzing the system with the system state information received from monitoring module. Analysis model, which comprises goal models with utility functions and state model, is exploited for analysis. Information required for diagnosis is passed to diagnosis module if abnormal state of the managed system is detected.

3) Diagnosis Module
Diagnosis Module finds out the location of the defect which causes abnormal state. Diagnosis model, which includes fault model, is utilized to identify the system fault. The fault information found in diagnosis module is sent to planning module.

4) Planning Module
Planning module determines a course of action to adapt by using policy model. Fault information received from the diagnosis module is used to extract proper policies in policy model. Planning module selects the most appropriate policy from the extracted policies and then determines the execution order of the selected policy.

5) Execution Module
Execution module performs the actions included in the selected policy by assigning the adaptation actions to effectors in the managed CPS elements.

6) Evaluation Module
Evaluation module checks whether the adaptation is successful and stores the adaptation result in the knowledge base. The accumulated adaptation results are exploited for feedback.

7) Knowledge Base
Knowledge base provides models to other modules and stores adaptation results. The models generated by developers in design time are maintained by autonomic computing engine at runtime.

8) Communication Handler
Communication handler reduces communication overhead between autonomic computing engine and managed CPS elements by using the thread pooling technique.

B. Operation Process

To enable the system to adapt for system failure and environment change, the system senses system state and environment and takes appropriate action. Fig. 2 shows overall operation process and data flow between the modules in the autonomic computing engine. Sensor inserted in CPS element is requested to sense the system state by monitoring module when 1. the time quantum is periodically expired, 2. the predefined check point is reached or 3. the execution of specific function is terminated. To acquire adequate system state information effectively, analysis module provides the lists which include information needed to analyze the system state through monitoring module.

After obtaining the system state information, analysis module starts to analyze whether the system satisfies the system goal while the managed system is executing its own process. Analysis model which includes goal model with utility function and state model is exploited to examine goal satisfaction level. Unlike other approach [7,8], our system only checks the goal violation in this step so that overhead of monitoring and analysis is reduced. When goal violation is not detected, the managed system finishes the adaptation process and continues its original process.

When goal violation is detected, analysis module sends the violated goal ID to diagnosis module. Diagnosis module starts to find out the root cause by using diagnosis model which includes fault model. Traditional fault models are used only in design time. However, our proposed fault model can be used at runtime for identifying the root cause for the system failure. To find out the root cause by using the fault...
Figure 2. Operation process of the proposed framework model, additional system state information is collected by sensor through the monitoring module. After identifying the root cause, root cause ID is sent to planning module.

Planning module receives root cause ID and extracts the policies concerned with the root cause ID in knowledge base. To select the most appropriate policy within the extracted policies, planning module uses policy model. Policy model is represented by policy weight table. Initial weight of each policy is assigned by developers in design time and modified at runtime through feedback from evaluation module. Planning module selects the policy with the highest value and delivers the policy ID to execution modules. Execution module extracts the execution IDs related to policy ID received from planning module and orders effectors in CPS elements to execute the adaptation actions related to the execution IDs. Effector executes the adaptation code related to the execution ID. Adaptation codes in CPS elements are inserted at design time or runtime. If the adaptation code to be executed does not exist in effector, effector requests the code to execution module. After requested, execution module compiles the code in the knowledge base and transfers it to effector.

After an adaptation process is terminated, evaluation module gathers the adaptation result. The result includes policy ID, action IDs, and relative level of goal satisfaction. These values are stored in the knowledge and utilized for planning module to modify weights of policies for feedback.

Figure 3. Development process of the proposed framework

C. Development Process

To develop the autonomic CPS without significant efforts, we provide automated design tools which contain goal graph editing tool, fault tree editing tool, monitoring and adaptation template code generator. First, developers create the goal model with goal graph editing tool. Goal graph is represented by tree-like structure. The root node of goal graph means the system root goal which the managed system must achieve finally. The leaf node of goal graph means the atomic goal that cannot be separated into other goals. The leaf goals are mapped to atomic function of the managed CPS.

Generally goal model is completed at requirement analysis step. However, in our approach, goal graph is modified through the whole development process of the managed CPS. If goal model is fixed, developers start to write the fault model. Fault model consists of several fault trees. The root node of each fault tree is mapped into the violation of leaf goal in goal graph. Developers create the policy table and assign the weight after analyzing possibilities of each fault. Adaptation actions are mapped to each policy. Adaptation actions can be duplicated if needed. Monitoring codes and adaptation codes are produced by filling the template codes that are generated at the previous step. The generated codes are compiled with the system code to complete the whole system.

IV. CASE STUDY: HOME SURVEILLANCE ROBOT

To demonstrate the feasibility of our approach, we implemented the autonomic computing engine and the editing tools in JAVA, but we have been developing it in C++ as well. To validate the usefulness, we apply our framework to Home Surveillance Robot scenario.

A. Scenario

Home Surveillance Robot is devised to help a physically handicapped person. The robot moves to a place directed by the person and then shots and transfer a picture to the monitor near the person. Occasionally robot cannot achieve the order because the environment is not normal, for example, there is an obstacle on the path that the robot follows.

B. Experimental Result

At First, we created the analysis model, which included the state model and the goal model. Fig. 4 shows the detail. We then mapped leaf goals to each state. This mapping information was stored into knowledge base. After completing the analysis model, we prepared several fault trees, one of which is illustrated in Fig.5. Each node needed the certain...
information to find out the root cause, so we mapped this information to monitoring module. We then created policy model and assigned the weight. Finally, all models prepared, we completed the robot by filling the template code and implementing the system process code.

**Case 1: Normal Environment**

In this case, we focused on measuring the overhead which inevitably occurred in monitoring and analysis. This generally blocks the induction of autonomic computing. In comparison with the other approaches [7,8], our approach needs less information because we separate determining whether the system state is normal or not and finding out the root cause. If the system state is determined to be normal, diagnosis is not triggered. Therefore, in normal state, the overhead of monitoring and analysis can be reduced.

**Case 2: Abnormal Environment**

The environment of the system was changed while the system was processing its own process. This change caused the failure of achieving the system goal. When the robot was blocked by the obstacle on the path, the violation of the goal, which means the robot did not arrive within a certain time, was detected by analysis module. After finding out the root cause and performing adaptation actions, the robot found the new path and moved through it. In this experiment, we discovered that our approach could empower the system to change its behavior according to the environment changes.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed the autonomic computing framework for CPS including the architecture, operation process and development process. The CPS managed by autonomic computing engine can enhance their ability to adapt to changes in environment and system failure. To validate our approach, we have provided Home Surveillance Robot scenario. We confirmed the validity of our proposed approach through case study.

For the future work we intend to develop our framework to reduce development overhead and to enhance the efficiency of the system. We are currently developingquadroter by applying our approach. We will apply our approach to several systems to validate our framework.

ACKNOWLEDGMENT

This work was supported by the IT R&D Program of MKE/KEIT[10035708, “The Development of CPS(Cyber-Physical Systems) Core Technologies for High Confidential Autonomic Control Software”]

REFERENCES