Home Network Service Programs described in a Rule-based Language

Muneyuki Shimokura Shuichi Nakanishi Tadashi Ohta

Information Systems Science
SOKA University
ohta@t.soka.ac.jp

Abstract— Rule-based languages allow programs to be easily described at a high and abstract level, however, they suffer from long execution times. A state transition model, one of a standard model for developing telecommunication services programs, can be used to simply described home network services. To reduce the complexity of development, this paper proposes a method for describing service programs using a rule-based language. In this paper the easy-of-use of this method for novice users is tested using a home network to control a robot toy.

Keywords-software architecture; rule-based language; Home network

I. INTRODUCTION

Much man-power is required to modify existing programs. To modify a program described in a procedural language, correct understanding source codes of the program is strictly required. So, there are many cases where modifying the existing program requires more man-power than developing a program as a completely new one.

For a rule-based language, the execution order of codes has nothing to do with the order of describing codes. Hence it is said that it is easy to add or modify a program described in a rule-based language. However, in general, an interpreter program is required to execute programs described in a rule-based language. So, execution efficiency drops greatly, compared to a case where programs are described in a procedural language [1].

Many IP services such as Home network services can be described in the state transition model. State transition conditions can be naturally described using a rule-based language. This paper proposes a method for describing Home network service programs using a rule-based language. In the proposed method, a software structure that keeps a decline in the program efficiency at reasonable rebel and provides a merit in adding or/and modifying functionalities to the existing programs.

To evaluate easiness of describing programs and the decline in the program efficiency, a Home network service software system that controls robots was experimentally developed. Evaluations show that the proposed method is effective and reasonable.

II. PROBLEMS AND SOLUTIONS

A. Problems

(1) A decline in the program efficiency

Programs described in a rule-based language are interpreted by a interpreter program to execute. Procedure in the interpreter, in general, requires so much time. Eventually, it is said that the program efficiency drops drastically compared to the programs described in a procedural language. Hence, there are few cases where a rule-based language is used for describing programs, except for a particular field such as AI. In the field of telecommunication services, there are many cases where service specifications are described using a rule-based language. Especially to detect feature interactions automatically, in many proposed methods, service specifications are described in a rule based-language [2-4]. But for service programs that are installed in service servers and executed to provide actual telecommunication services, as far as the authors know, there are no cases except for the authors work [5].

To adopt a rule-based language for describing Home network services programs, the decline in the program efficiency should be prevented.

(2) Processing model in Home network services

The state transition model has achieved satisfactory results in the processing model for telephone services. Though, there is a case where the state transition model is used for describing specifications of Home network services [6], as far as the authors know, there are no cases where the state transition model is used for describing service programs for Home network services. In telephone services, supplementary services have been developed by adding functions to the basic

International Conference on Software Engineering Advances(ICSEA 2007)
0-7695-2937-2/07 $25.00 © 2007 IEEE

service (POTS; Plain Old Telephone Service) [7]. But there is not a basic service in Home network services, such as POTS.

Thus, it is required to provide the new software architecture for Home network services to which the state transition model can be applied so that new Home network services can be developed easily.

B. Solutions

(1) A decline in the program efficiency

A service program consists of two parts: a service scenario description part and a part that describes individual procedures to realize the service scenario. Individual procedures are such as requirement analysis, devise selection, device control and datum access, and can be described as independent programs. Service scenarios represent flows of procedures for each service; the order of executions of procedures described above.

The most difficult work in developing service programs is to describe the service scenario. So, by using a rule-based language to describe the service scenario, it is expected that service programs can be easily described and adding/modifying functionalities can be easily done. Individual procedures can be developed as program parts to be re-used for many program systems. So, to make a decline in the program efficiency small, the individual procedure parts are described in a procedural language. Interfaces to scenario parts, which are described in a rule-based language, are provided as names of commands. By setting a command description part in the rule-based language, the individual procedures are triggered.

(2) Processing model in Home network services

As shown in [6], many Home network services can be developed by orchestrating some services. So, a layered state transition model is proposed. In a lower layer, state transition diagrams for individual services are described. In a upper layer, scenarios which orchestrate the individual services are described as state transition diagrams of integrated services. Service scenarios in each layer are described in a rule-based language according to individual state transition diagrams.

III. SOFTWARE ARCHITECTURE STAR

A brief explanation on STAR (SofTware Architecture using a Rule-based language), which is a software architecture adopted in this paper, is given. For more detailed explanation, please refer to [5,8].

A. System State

A system state of a service server is represented as a set of states of devices or relationships between devices called 'primitive'. In some services, states of human or environment such as temperature, humidity, brightness, and so on are also treated. Arguments in a primitive represent a device, human, and environment, respectively.

Identify applicable sponsor/s here. *(sponsors)*

Suppose, terminal A is calling terminal B (denoted by call(A,B)) and terminal C and D are in talking state (denoted by talk(C,D)). In this case, the system state, to which terminal A, B, C and D are connected, is represented as {call(A,B),talk(C,D)} (Figure 3).

B. ESTR

ESTR is a rule-based language developed to be used in the STAR. ESTR has the form of Pre-condition, event, Post-condition and Action-description. See Figure 1.

Pre-condition Event: Post-condition {Action description}

Fig. 1 A form of ESTR

Pre-condition shows a condition of the rule to be applied and is represented as a set of primitives. An event is a trigger that causes the state transition. Post-condition shows a condition of a system state after the rule is applied and is also represented as a set of primitives. Action-description is the system control description part that shows the system controls required for the state transition. When no system controls are required, the content of {} is empty. A description example of ESTR is shown in Figure 2.

call(x,y) connotify(y,x): talk(x,y),{Send(con,y,x),Con(x,y)}

Fig. 2 An Example of ESTR

The example in Figure 2 is explained. Terminal x is calling terminal y, denoted by call(x,y). If terminal y makes off hook, denoted by connotify(y,x), a signal Connect is sent to terminal x, denoted by Send(con,y,x), and terminal x and y transit to talk state, denoted by talk(x,y). call(x,y) and talk(x,y) are primitives. All arguments in primitives are described as variables so that a rule can be applied to any terminals.

When an event occurs, a rule which has the same event and whose Pre-condition is included in the system state is applied. When the rule is applied, stored programs designated by Action-description are executed. When the programs end normally, the system state changes as follows. A state corresponding to the Pre-condition of the applied rule is deleted from the current system state and a state corresponding to Post-condition of the applied rule is added. Here, a state corresponding Pre/Post-condition is obtained by replacing arguments in Pre/Post-condition with actual terminals when the rule is applied.

Suppose, the system state is {call(A,B),talk(C,D)}. At this moment, an event connotify(B,A) occurs. The rule shown in Figure 2 has an event connotify(y,x). Replace variables x and y with A and B, respectively. Then the Pre-condition of the rule, call(A,B), is included in the system sate. So, the rule is applied. Then, the system state changes into {talk(A,B),talk(C,D)}. This state transition diagram is shown in Figure 3.

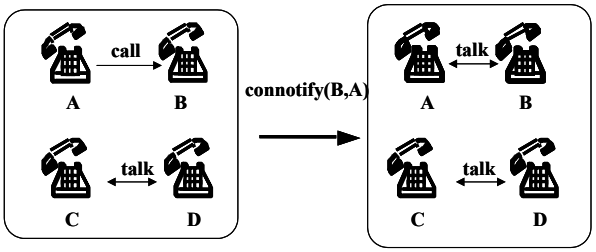


Fig. 3 A state transition diagram

C. *Execution Environment (EE)*

For STAR, the EE consists of: An ESTR Interpreter, which selects a rule and executes the selected rule, an Input processing part, which receives input signals from a platform provided by a vender and converts them into events, and an Action executing part, which analyzes and executes the Action-description of the rule. The software architecture of the EE of STAR is shown in Figure 4.

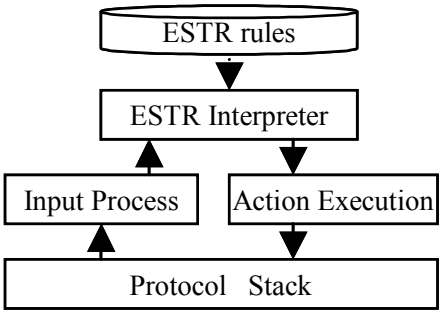


Fig. 4 Software Structure of EE

1) Input Processing Part

When a signal is received, the Input processing part converts the signal to an event corresponding to the signal, so that the ESTR Interpreter can handle the event. The event is sent to the interpreter.

2) ESTR Interpreter

On receiving an event from the Input processing part, the interpreter selects a rule which has the same event as one sent.

When the rule to be applied is selected, the Action-description of the rule is sent to the Action executing part. If the execution in the Action executing part ends normally, the system state is changed according to the Pre- and Post-conditions of the selected rule. Otherwise, the system state is not changed.

3) Action executing part

On receiving an Action description from the interpreter, the Action executing part of the program analyzes it and decides which programs are to be executed and in what order. The programs have been stored in the system beforehand.

IV. STATE TRANSITION DIAGRAM

A. *A layered State Transition Diagrams*

In the lowest layer, state diagrams which are used as elements for a single service are described. In the second layer, state transition diagrams for each single service are described, calling state transition diagrams in the lower layer if needed. In the third layer, state transition diagrams for integrated services are described, calling state transition diagrams in the lower layers if needed. In the upper layer there may be state transition diagrams for integrated services which orchestrate other integrated services in the lower layer.

A concept of layered state transition diagram is shown in Figure 5.

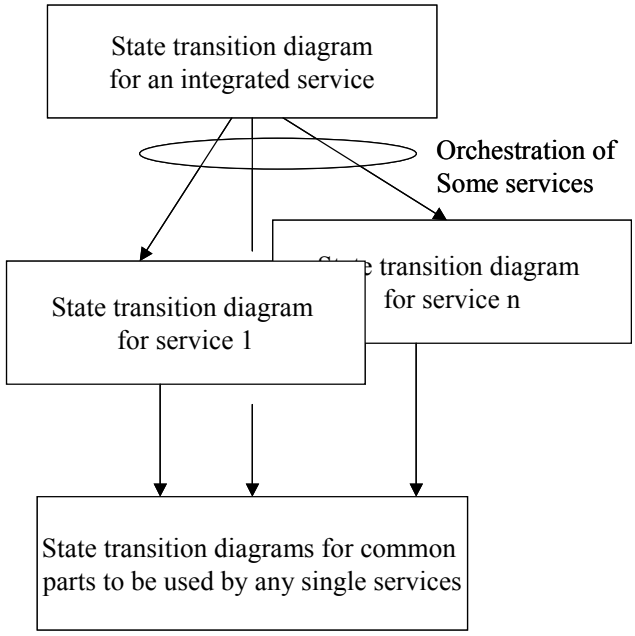


Fig. 5 Layered state transition diagram

B. *Separation of State Space*

For Home network services, generally, devices have complex functions. Besides, in many cases, each functions can have states independently. Thus, if states of a Home network service is represented in one state space, the number of states becomes huge. Huge number of states causes increase in the number of rules. This introduces program bugs and increase in processing time for the interpreter program. So, it is proposed that service states are separated based on functions which can work independently.

A concrete example for robot control system is explained. For a robot, functions of body (go forward, go backward, stop, turn, circle round), head (up, down, right, left, round), and mouth and ears (speak, listen) can work independently. Suppose these three functions have n1, n2 and n3 states, respectively. If states of the robot are represented in one state

space, there are n1×n2×n3 states as service states. But if the state space is separated for each functions, the total number of states is n1+n2+n3. In general, n1+n2+n3 << n1×n2×n3.

V. Application To Robot Control System

To evaluate solutions proposed in *B* in section II and in section IV, experimental system, which control robots in a home or at office, was implemented.

A. Implementation

(1) System configuration

Figure 6 shows an experimental system. A robot used in the system called AIBO which is made by Sony and the most famous toy robot in Japan.

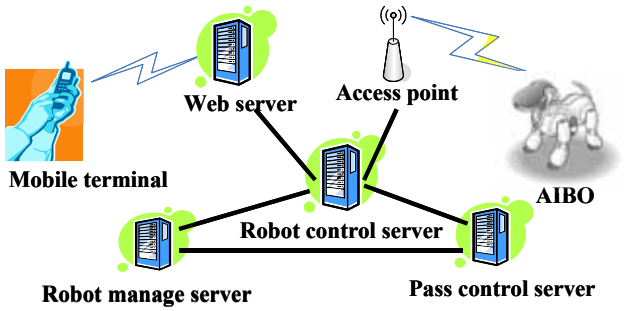


Fig. 6 Experimental System

a) Robot control server

This server controls robots communicating with other servers.

b) Robot manage server

This server manages robots. For requirements from the robot control server, selects an appropriate robot.

c) Pass control server

This server selects an appropriate pass to reach the goal. In many cases, it selects the shortest pass between given spots.

d) Web server

This server communicates servers described above with mobile terminals and other web servers via the Internet and LAN. Mobile terminals are used by users as interface devises.

(2) Services

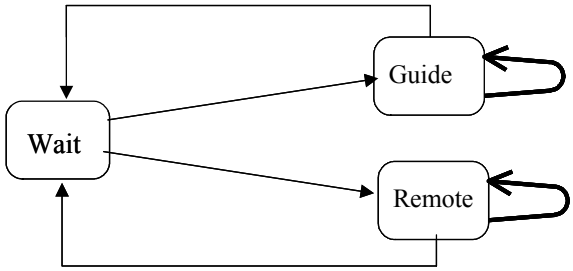
a) Remote control

First of all, to develop services commonly used for many services, a remote control service which using mobile terminals move robots were implemented. According to instructions input by a mobile telephone or a PDA, robots move forward, back ward, stop, turn, and even dance. The robots move head and read aloud sentences sent from the robot control server.

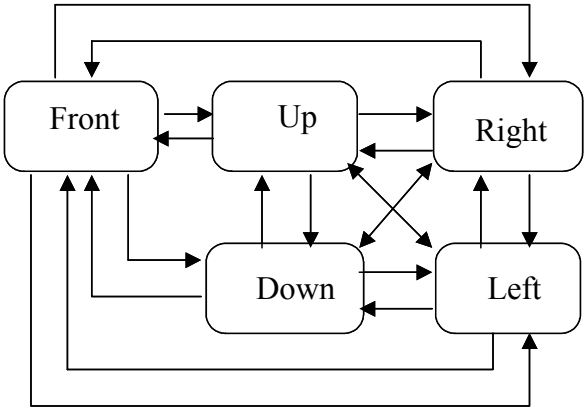
b) Guide service

When a visitor reaches entrance, a robot goes to the entrance to meet him/her. After greeting, the robot lead the visitor to the destination.

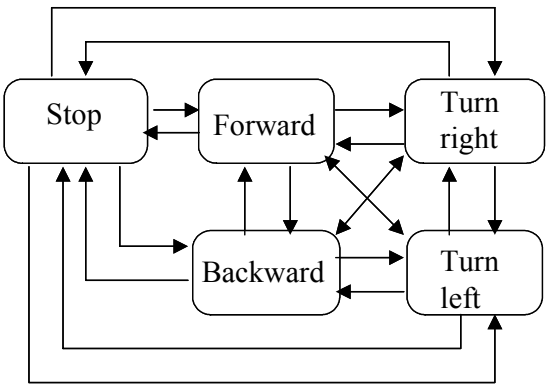
(3) State transition diagrams and ESTR rules



(1) State transition diagram for integrated service



(2) State transition diagram for face movement



(3) Part of state transition diagram for body movement

Fig. 7 State transition diagram

State transition diagrams for services described in (2) are partially shown in Figure 7.

A set of ESTR rules for remote control service is described below. In some rules, as a body movement, a word 'circle' appears. This means circle round from the left or right.

rule[0]: stop(x) forward(x):
forwarding(x) {Robot_Action(FORWARD,x)}
rule[1]: stop(x) backward(x):
backwarding(x) {Robot_Action(BACKWARD,x)}
rule[2]: stop(x) turn_left(x):
turning_left(x) {Robot_Action(TURN_LEFT,x)}
rule[3]: stop(x) turn_right(x):
turning_right(x) {Robot_Action(TURN_RIGHT,x)}
rule[4]: stop(x) circle_right(x):
circle_right(x) {Robot_Action(CIRCLE_RIGHT,x)}
rule[5]: stop(x) circle_left(x):
circle_left(x) {Robot_Action(CIRCLE_LEFT,x)}
rule[6]: forwarding(x) stop(x):
stop(x) {Robot_Action(STOP,x)}
rule[7]: forwarding(x) turn_left(x):
turning_left(x) {Robot_Action(TURN_LEFT,x)}
rule[8]: forwarding(x) turn_right(x):
turning_right(x) {Robot_Action(TURN_RIGHT,x)}
rule[9]: forwarding(x) backward(x):
backwarding(x) {Robot_Action(BACKWARD,x)}
rule[10]: forwarding(x) circle_right(x):
circle_right(x) {Robot_Action(CIRCLE_RIGHT,x)}
rule[11]: forwarding(x) circle_left(x):
circle_left(x) {Robot_Action(CIRCLE_LEFT,x)}
rule[12]: turning_right(x) stop(x):
stop(x) {Robot_Action(STOP,x)}
rule[13]: turning_right(x) turn_left(x):
turning_left(x) {Robot_Action(TURN_LEFT,x)}
rule[14]: turning_right(x) forward(x):
forwarding(x) {Robot_Action(FORWARD,x)}
rule[15]: turning_right(x) backward(x):
backwarding(x) {Robot_Action(BACKWARD,x)}
rule[16]: turning_right(x) circle_right(x):
circle_right(x) {Robot_Action(CIRCLE_RIGHT,x)}
rule[17]: turning_right(x) circle_left(x):
circle_left(x) {Robot_Action(CIRCLE_LEFT,x)}
rule[18]: turning_left(x) stop(x):
stop(x) {Robot_Action(STOP,x)}
rule[19]: turning_left(x) forward(x):
forwarding(x) {Robot_Action(FORWARD,x)}
rule[20]: turning_left(x) turn_right(x):
turning_right(x) {Robot_Action(TURN_RIGHT,x)}
rule[21]: turning_left(x) backward(x):
backwarding(x) {Robot_Action(BACKWARD,x)}
rule[22]: turning_left(x) circle_right(x):
circle_right(x) {Robot_Action(CIRCLE_RIGHT,x)}
rule[23]: turning_left(x) circle_left(x):
circle_left(x) {Robot_Action(CIRCLE_LEFT,x)}
rule[24]: backwarding(x) stop(x):
stop(x) {Robot_Action(STOP,x)}
rule[25]: backwarding(x) turn_left(x):
turning_left(x) {Robot_Action(TURN_LEFT,x)}
rule[26]: backwarding(x) forward(x):
forwarding(x) {Robot_Action(FORWARD,x)}
rule[27]: backwarding(x) turn_right(x):
turning_right(x) {Robot_Action(TURN_RIGHT,x)}
rule[28]: backwarding(x) circle_right(x):
circle_right(x) {Robot_Action(CIRCLE_RIGHT,x)}
rule[29]: backwarding(x) circle_left(x):
circle_left(x) {Robot_Action(CIRCLE_LEFT,x)}
rule[30]: face_front(x) face_up(x):
face_up(x) {Robot_Action(FACE_UP,x)}
rule[31]: face_front(x) face_down(x):
face_down(x) {Robot_Action(FACE_DOWN,x)}
rule[32]: face_front(x) face_left(x):
face_left(x) {Robot_Action(FACE_LEFT,x)}
rule[33]: face_front(x) face_right(x):
face_right(x) {Robot_Action(FACE_RIGHT,x)}
rule[34]: face_up(x) face_front(x):
face_front(x) {Robot_Action(FACE_FRONT,x)}
rule[35]: face_up(x) face_down(x):
face_down(x) {Robot_Action(FACE_DOWN,x)}
rule[36]: face_up(x) face_left(x):
face_left(x) {Robot_Action(FACE_LEFT,x)}
rule[37]: face_up(x) face_right(x):
face_right(x) {Robot_Action(FACE_RIGHT,x)}
rule[38]: face_down(x) face_front(x):
face_front(x) {Robot_Action(FACE_FRONT,x)}
rule[39]: face_down(x) face_up(x):
face_up(x) {Robot_Action(FACE_UP,x)}
rule[40]: face_down(x) face_left(x):
face_left(x) {Robot_Action(FACE_LEFT,x)}
rule[41]: face_down(x) face_right(x):
face_right(x) {Robot_Action(FACE_RIGHT,x)}
rule[42]: face_left(x) face_front(x):
face_front(x) {Robot_Action(FACE_FRONT,x)}
rule[43]: face_left(x) face_up(x):
face_up(x) {Robot_Action(FACE_UP,x)}
rule[44]: face_left(x) face_down(x):
face_down(x) {Robot_Action(FACE_DOWN,x)}
rule[45]: face_left(x) face_right(x):
face_right(x) {Robot_Action(FACE_RIGHT,x)}
rule[46]: face_right(x) face_front(x):
face_front(x) {Robot_Action(FACE_FRONT,x)}
rule[47]: face_right(x) face_up(x):
face_up(x) {Robot_Action(FACE_UP,x)}
rule[48]: face_right(x) face_down(x):
face_down(x) {Robot_Action(FACE_DOWN,x)}
rule[49]: face_right(x) face_left(x):
face_left(x) {Robot_Action(FACE_LEFT,x)}

B. Evaluations

(1) A decline in the program efficiency

Program efficiency can be kept about 50 % down compared to the program all of which were described in C language. For reasons why the program efficiency doesn't

drop so much, the following two reasons can be considered: First, when service programs are described in the state transition model, scenario description part corresponds to the program which decides the task to be executed. This part, even if described in C, requires much processing time. Second, the language specification of ESTR is so simple that ESTR interpreter does not require so much processing time.

(2) Ease in describing ESTR

To evaluate ease in describing ESTR, 19 high school students, who are 16 - 17 years old and have no experiences for programming, developed a remote control system program using ESTR. They had a half hour lecture on ESTR. Next, they learned how to operate PC, to use editor, and to install ESTR rules in the robot control server for one hour. Then they tried to describe the program in ESTR. All students can describe the program and install the program to the server and enjoyed to control robots.

The number of ESTR rules described by students is different. Average number is around 50. Most students could develop the program in one hour. In the shortest case, it took 40 minutes. In the longest case, it took one and half hours. Thus, ease in programming using ESTR was confirmed.

VI. CONCLUSIONS

To develop service programs for Home network easily, a method for describing service programs using a rule-based language was proposed. Based on the state transition model, network robot service programs were developed and effectiveness of the proposed method was confirmed.

As future work, many network robot services to be used in home and office will be developed.

REFERENCES

[1] Ian Sommerville, "Software Engineering," 1991. Addison-Wisley

[2] "FEATURE INTERACTIONS IN TELECOMMUNICATIONS AND SOFTWARE SYSTEMS II - VII," Proc. of FIW, IOS Press.

[3] "FEATURE INTERACTIONS IN TELECOMMUNICATIONS AND SOFTWARE SYSTEMS II - VII," Proc. of ICFI05, IOS Press.

[4] Gilmore and Ryan Eds, "Language Constructs for Describing Features," Proc. of FIREworks, Jun, 2000, Springer.

[5] T. Morinaga, G. Ogose, and T. Ohta, "Active Networks Architecture for VOIP Gateway Using Declarative Language," IEICE Trans. on Communications, Vol. E84-B, No.12, pp.3189-3197, Dec. 2001.

[6] M. Nakamura, H. Igaki, and K. Matsumoto, "Feature Interactions in Integrated Services of Networked Home Appliances: An Object-Oriented Approach," Proc. of ICFI05, pp.236-251, Jun 2005. IOS Press.

[7] N. Grifeth, R. Blumenthal, J. Gregoire, and T. Ohta, "Feature Interaction detection contest of the Fifth International Workshop on Feature Interactions," Computer Networks Vol. 32, pp.487-510, Apr. 2000.

[8] T. Yoneda and T. Ohta, "The declarative language STR (State Transition Rules)," Proc. of FIREworks workshop, pp.197-212, Jun 2000. Springer.

0-7695-2937-2/07 $25.00 © 2007 IEEE