IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 2, February 2015.
www.ijiset.com
ISSN 2348 – 7968

Dynamic Rule Modification Method for Rete-ECA Algorithm

Rachel Lee[1] and Sang-Young Cho[1]

[1] Department of Computer Engineering, Hankuk University of Foreign Studies,
Yongin, Gyeongi, Korea

Abstract

Context-aware services enable users to interact with their environment in more useful ways. Rule-based systems are used for base vehicles for context-aware services due to their powerful capability to represent situations and corresponding actions to be performed. Usually, environments where context-aware services are applied are not static but have somewhat dynamic features. This paper describes a design and implementation of dynamic rule modification method for a Rete-based rule inference engine called Rete-ECA for device control and management. The experiment shows that the rule modification method can be used to accelerate execution when the rule set is complex and more than one device variables are related to rule conditions.

Keywords: *Context Aware Service, Rule Inference Engine, Rete Algorithm, Dynamic Environment, Rule Modification.*

1. Introduction

Context-aware services enable users to interact with their environment in more useful ways. Applications can use context-aware services to determine which options should be made available to the user with respect to the current context. Although there is some debate about the precise meaning of context [1], in this paper, it refers to information about a user's surroundings, location, and/or preferences. This information may include the user's location, as well as the devices available to the user at that location. As the context information changes, the user may move to a different location or change her/his preferences, the context-aware service must be able to update the context information that it is using and change which additional services are available.

These additional services may become available through connections to nearby devices. This requires that the context-aware service have some means of managing and controlling these devices. Using a context-aware service to determine how to incorporate context information into the operation of applications provides a quick and easy way context-aware control. Rule-based systems can be used to achieve this end [2]. The designer first writes a rule set that specifies how the devices and application should interact. The rule set along with context information are then fed into an inference engine, which interprets which rules are relevant at the current time and finds actions that are appropriate for the system or application to take.

Usually, environments where context-aware services are applied are not static but have somewhat dynamic features. The devices surrounding users may be moved to other locations or the characteristics of them may be changed. This means that a set of rules to describe an environment situation can be changed over times. This paper describes a design and implementation of dynamic rule modification methods for a Rete-based rule inference engine called Rete-ECA for device control and management [3].

The Rete-ECA system is a Rete-based rule network for storing rule and device information. This information can be used for determining appropriate actions for the system to take given the current state of the environment. The method presented here operates in accordance with the Event-Condition-Action (ECA) model, which means that when an Event occurs, the system should check certain Conditions, and then fire the appropriate Actions if the Conditions are true. This system explicitly uses its events to determine when and which portions of the rule engine should activate at a specific time and thus consumes less time than the original Rete algorithm [4].

This paper shows that the dynamic rule modification methods suggested in this paper can save memory consumption and execution time during rule matching operations.

In section 2, related work in rule matching algorithms and their use in context-aware environments is discussed. In section 3, the proposed dynamic rule modification methods are presented. Experiment results are showed and discussed in section 4. Finally, this paper concludes with section 5.

2. Previous Work

Rule-based systems are useful for problems that can benefit from using rule sets to determine how appropriate solutions can be found. Rule-based systems capture,

ISSN 2348 – 7968

represent, store, distribute, reason about, and apply human knowledge using causal if-then reasoning to produce some result. This result can be an answer to a question, a solution to a problem, a data analysis [2], or in the case of Rete-ECA, a sequence of actions that should be taken.

The Rete match algorithm [4] is one of the most popular many pattern/many object pattern matching algorithms. The Rete match algorithm uses two main structures. The first is a working memory, a set of elements that represent facts that enter the system. The working memory can be said to provide a model of the states of each object in the system. The second major component of this system is the pattern matching network. The structure of this network is determined by the condition portion of all the rules that the system uses. The pattern matching network is itself a directed acyclic graph of nodes and is divided into the alpha network and the beta network [5]. The alpha network is used to perform simple tests of condition parts and the beta network is used for join the results of the alpha network. For example, consider the rule "SendAlarm" given by

```
rule SendAlarm
    if
      q : Earthquake( magnitude  >  5 )
      h : House( address  = q.location )
   then
      sendAlarmTo( h.owner )
   end
```

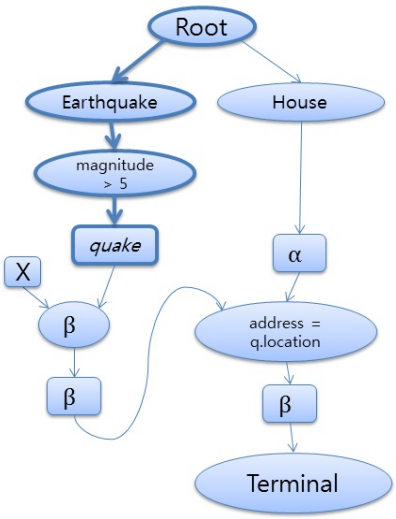

Fig. 1 Rete rule network example

The structure created in the Rete pattern matching network is shown in Figure 1. Circles are used to represent nodes while rectangles are used to represent memories. Null input is indicated with "X". There are two type nodes, one that checks for the object type Earthquake and the other that checks for the object type House. The alpha condition, "magnitude > 5", is stored at the alpha node, which is connected to an alpha memory. The bold arrows indicate the path through the network that the working memory element has made. Because quake's working memory element has passed its alpha tests, it is stored in an alpha memory.

The bottom or right side of the network (beta network) is made up of two-input nodes. They combine the working memory elements stored in the memories at the end of each chain of one-input nodes, joining two paths of the network into one path. For ease of implementation, alpha memories may be fed into a dummy beta node with a null left input.
The two-input nodes, also called beta nodes, contain inter-element tests. The inter-element tests check if the facts associated with two working memory elements are compatible. One working memory element is stored in the beta (or alpha, depending on the implementation) memory from the left path while the other is stored in the alpha memory from the right path.

The Rete match algorithm stores element data at each alpha memory and each beta memory. The size of the beta memories can increase exponentially. TREAT addresses this issue and eliminates the memory requirement in the beta network, but at the cost of increased processing time and a more complex memory in the alpha network [6]. LEAPS uses a lazy matching algorithm to trade some completeness for much higher performance [7].

There are some research results to utilize rule engines for context-aware services. Rete-ADH (Alpha Network Dual Hashing) [8] is an optimization of the Rete match algorithm for context-aware services. Rete-ADH optimized the alpha network using the dual-hashing technique. Rete-ADH selects facts fast and reduces the size of the beta memories. MiRE (Minimal Rule Engine) [9] was devised as a lightweight, context-aware rule-based system for resource-limited cellular phones. MiRE uses a modified Rete network for rule-matching. Rete-ECA [3] is an inference engine for context-aware device control and management.

3. Dynamic Rule Modification for Rete-ECA

The main purpose of Rete-ECA is to provide a platform for smaller devices with higher constraints on the amount of memory and processing time devices in a dynamic

ISSN 2348 – 7968

control environment. To achieve the purpose, the inference engine avoids wasteful polling by executing only after an event has occurred. The performance is further enhanced by dividing a rule network into two rule and device networks and avoids unnecessary rule checking. In addition, the rules can be adjusted dynamically. In this section, we focus on the dynamic rule modification methods for efficient execution in dynamically varying situations.

3.1 The Rule Network Structure of Rete-ECA

The Rete-ECA rule engine consists of two main components: the rule network and the device network. Both the rule network and device network are Rete networks. The main difference between the rule network and the device network is that the sequence of conditions that make up the left-hand side of each rule are used to build the rule network, whereas the sequence of device conditions that make up a portion of the right-hand side of the rule are used when creating the device network. Working memory elements that refer to users in the environment and objects that contain information specifically about the state of the environment enter the rule network only.

For example, the rule "LeftLightOn" is described in plain English as

If the mouse's previous location was P and its current location is S, the trial mode is 0, the left blue light is off, and the right blue light is off, then, if the left blue light, the speaker, and left feeder is enabled, then turn on the left blue light and enable the devices that may need to be used to perform the actions of the other rules.

The rule, "LeftLightOn" translates to

PSMouse:MOUSE(PastLocation = "P", CurrentLocation = "S")
ExpInfo:ExperimentInfo(trialMode = 0)
RightLight:BLUELIGHT(IsOn = False, DID = 1)
LeftLight:BLUELIGHT(IsOn = False, DID = 0)

where MOUSE, ExperimentInfo, and BLUELIGHT refer to object types. PSMouse is the name given to the MOUSE object with a PastLocation of "P" and a CurrentLocation of "S". ExpInfo is the name of the ExperimentInfo object with a trialMode variable whose value is 0. RightLight is the object of type BLUELIGHT that is not turned on and whose device ID (DID) is 1. Likewise, LeftLight is the name given to the BLUELIGHT object that is not turned on and whose device ID (DID) is 0.

The device conditions translate to the device rule, "LeftLightOn_ACTION," which is

LeftLight:BLUELIGHT(IsEnabled = True,DID = 0)
Buzzer:SPEAKER(IsEnabled = True,DID = 0)
LeftFeeder:FOOD(IsEnabled = True,DID = 0)

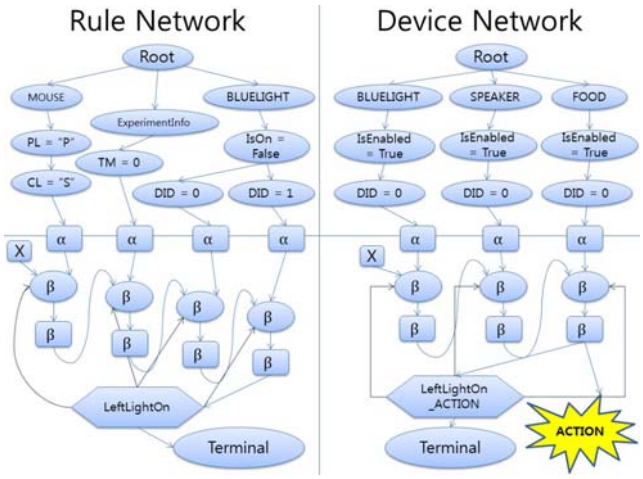

Fig. 2 Structure of the Rete-ECA rule inference engine with one rule, LeftLightOn.

Figure 2 illustrates the structure created to represent the LeftLightOn rule in the Rete-ECA inference engine after its initial creation.

3.2 Rule Network Modification

In an environment that is dynamically changing, a set of rules used may be changed. Some rules that are used situation by situation don't have to participate in whole rule matching processes. We devise rule activation/deactivation method to discard unnecessary rules in rule matching processes and it makes the Rete-ECA algorithm perform better than the algorithm without modification.

Rules in the rule network switch their status between "active" and "inactive" during rule activation/deactivation. A rule becomes active in the rule network once a group of devices becomes available to it. A group of devices becomes available when the corresponding device rule's final beta node has a reference to a group of devices stored in its memory. This occurs when all of the devices needed for executing the rule's actions have caused all of the device rule's conditions to be met. A rule becomes inactive when it no longer has any group of devices available to execute its actions. Only the alpha nodes of active rules are checked when searching for left-hand side pattern

ISSN 2348 – 7968

matches. This prevents the matching process from entering the beta network through alpha nodes that are used for matching inactive rules.

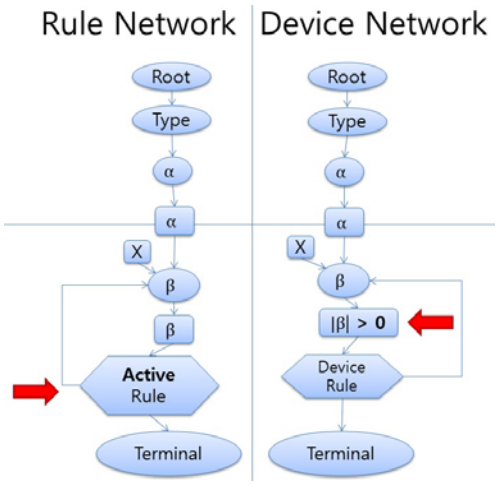

Fig. 3 Rule status changes in Rete-ECA with rule activation.

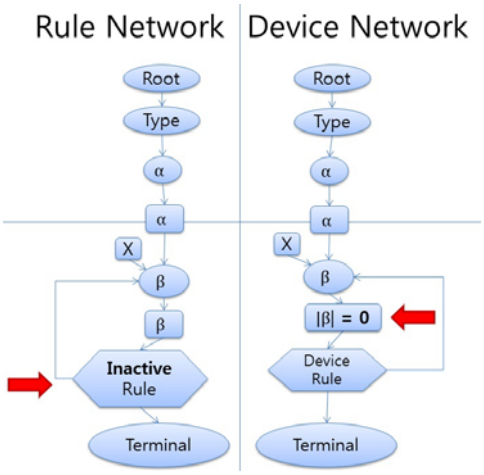

Fig. 4 Rule status changes in Rete-ECA with rule deactivation.

Rule activation/deactivation occurs when a change is made to the memory of the last beta node for a given sequence of device conditions in the device network. Figure 3 and Figure 4 illustrate the conditions for a rule's status change. This change is either the insertion or deletion of a group of device working memory elements. If this change causes the number of device groups in the last beta node's memory to increase, then the name of the rule is added to a list of rule names. This list indicates which rules will be activated during the next access of the inference engine. Likewise, if the change to the memory of the last beta node causes the number of device groups to drop to zero, the name of the rule is added to a list of rule names that will be deactivated during the next access to the inference engine.

In the next access to the inference engine, a working memory element is updated. This update requires that the old instance of the element is removed from the system and the updated version of this element reinserted. During the removal process, a pass is first made through the alpha network. If a rule must be activated or deactivated, then as the working memory element reaches these alpha nodes, the alpha nodes are updated if they are used by the rule being activated/deactivated. Once all of the occurrences of the working memory element are removed from the alpha network, the remaining nodes in the alpha network are then checked to see if they are used by the given rule and updated accordingly. Due to node-sharing in the alpha network and Rete-ECA's implementation, all chains of alpha nodes connected to the relevant type node must be checked and possibly updated.

Updating the given rule's status in the beta network is more straightforward, due to the implementation of Rete-ECA's beta network. Each rule stores references to the beta nodes that are used to check for the rule's object joins. If a rule's status changes between "active" and "inactive", its beta nodes are referenced and updated accordingly. During this process of simultaneous working memory element removal and rule activation/deactivation, (almost) the entire beta network must be accessed. This is due to the facts that active rules contain references to beta nodes that may contain groups of working memory elements that contain the given element and inactive rules that should become active must be accessed.

In addition, when a rule changes from "active" to "inactive," any memories it uses in the alpha or beta networks are flushed if no other active rules also used them.

4. Experiments

The Rete-ECA system described in the previous section was evaluated as the device control and management system for a mouse-tracking environment. Mice are placed in this environment and their behavior is observed and studied by psychology researchers. If the mice behave in a particular way in relation to the environment, they can be rewarded or punished. The rewards and punishments are determined by the researchers in a way that meets their research objectives. The environment (see Figure 5) used in these experiments is segmented into seven abstract locations called L, X, S, P, R, Y, and A. Transparent

ISSN 2348 – 7968

partitions are used to create physical barriers in the environment. There are two mouse pellet feeders, represented in the figure by the cheeses. There are two blue lights: one on the left side and one on the right side.

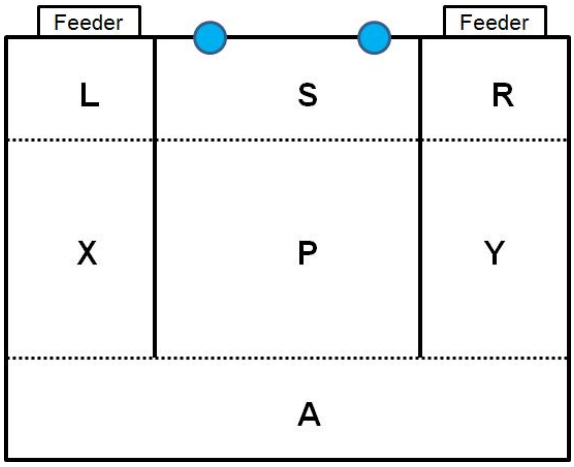

Figure 5 Diagram of the bottom level of the experimental environment.

The rule set used in this evaluation tracks one mouse for the duration of an experiment. In this evaluation, an experiment consists of 20 three-minute trials or lasts for 40 minutes, whichever occurs sooner. A trial is the amount of time it takes for the mouse to be rewarded or punished or three minutes, if these situations have not occurred. In the environment, the performance and size of the original Rete-ECA and Rete-ECA with rule activation/deactivation (Rete-ECA-AD) are presented and compared with twelve rules.

At first, we measured the program sizes of the two systems. The size of the original Rete-ECA system is 571KB and that of Rete-ECA-AD is 608KB that is slightly more than the original one due to the activation/deactivation code portion.

Secondary, we compared the amount of virtual memory consumption of two systems. The values are 9.98MB and 9.95MB for the original Rete-ECA and Rete-ECA-AD, respectively. Rete-based algorithms consume large memory to maintain rule networks and working memories. The run-time memory consumptions of two systems have no meaningful difference because Rete-ECA-AD maintains whole rules even though it does not use unnecessary rules.

Finally, we measured time in CPU clock ticks required for the matching process, rule network modifications, and action execution for each algorithm. The total number of clock ticks consumed by the evaluated systems averaged over three experiments. We used two types of rule sets: one is simple device condition (SDC) type and another is complex device condition (CDC) type. In the SDC rule set, each rule requires a group with only one device to execute its actions. The CDC rule set requires a group of one or more devices to execute its actions.

For SDC rule set, Rete-ECA and Rete-ECA-AD consume 4,195 and 4,361 clock ticks, respectively. In this case, rule modification is not effective because device on-off conditions have limited effect on the rule matching process. The rule modification overhead makes Rete-ECA-AD consume 104% clock ticks of Rete-ECA. For CDC case, Rete-ECA and Rete-ECA-AD consume 6,050 and 5,590 clock ticks, respectively. This result shows the Rete-ECA-AD system consumes 92% of the Rete-ECA clock ticks. Using rule modification, the matching process can be accelerated as the complexity of a rule set increases.

5. Conclusions

The focus of this paper is a design and implementation of a rule modification method for Rete-based inference engines. The original Rete algorithm consumes large memory to store intermediate matching results suffers from slow execution times. Many researches have undertaken to address the memory and performance issues. This paper proposed dynamic rule modification method that enables the Rete-based algorithm to enhance their performance in dynamically varying environments. The experiment shows that the rule modification method can be used to accelerate execution when the rule set is complex and more than one device variables are related to rule conditions.

In the future work, we will apply the Rete-ECA algorithm to extended problems in smart building or home automation control systems. The network partitioning technique developed here can be extended more partitions and will have many benefits for parallel hardware architectures.

Acknowledgments

This work was supported by Hankuk University of Foreign Studies Research Fund of 2014.

References

[1] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context Aware Computing for the Internet of Things: A

ISSN 2348 – 7968

Survey", IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, 2014, pp. 414-454.
[2] H.-R. Frederick, "Rule-based systems", Communications of the ACM, Vol. 28, No. 9., 1985, pp. 921-932.
[3] R. Lee and S.-Y. Cho, "Rete-ECA: A Rule-based System for Device Control", in Proc. of Int. Workshop on Artificial Neural Networks and Intelligent Information Processing, 2014, pp. 95-102.
[4] C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem", Artificial Intelligence, Vol. 19, No. 1., 1982, pp. 17-37.
[5] D. Sottara, P. Mello, and M. Proctor, "A Configurable Rete-oo Engine for Reasoning with Different Types of Imperfect Information", IEEE Trans. on Knowledge and Data Engineering, Vol. 22, No. 11, 2010, pp. 1535-1548.
[6] D. P. Miranker, "Treat: A Better Match Algorithm for AI Production Systems; long version", Technical report, University of Texas at Austin, 1987.
[7] D. Batory, "The leaps algorithm", Technical report, University of Texas at Austin, 1994.
[8] M. Kim, K. Lee, Y. Kim, T. Kim, Y. Lee, S. Cho, and C.-G. Lee, "Rete-ADH: An improvement to rete for composite context-aware service", Int. Journal of Distributed Sensor Networks, 2014, pp. 1-11.
[9] C. Choi, I. Park, S. J. Hyun, D. Lee, and D. H. Sim, "Mire: A minimal rule engine for context-aware mobile devices", 3rd Int. Conf. on Digital Information Management, IEEE, 2008, pp. 172-177.

Rachel Lee received his Bachelor's Degree in Computer Science from Swarthmore College, USA, in 2006, and M.S. degree in Computer Engineering from Hankuk University of Foreign Studies, Korea, in 2015. Her research is currently associated with rule-based system and system control.

Sang-Young Cho received his B.S. degree in Control and Instrumentation from Seoul National University, Korea, in 1988, and M.S. and Ph.D. degrees in Electrical Engineering from KAIST, Daejeon, Korea, in 1990 and 1994, respectively. He is currently a full professor with the Department of Computer Science and Engineering in Hankuk University of Foreign Studies. His research interests include embedded system, computer architecture, and software optimization.