Rete-ECA: A Rule-based System for Device Control

Rachel Lee and Sang-Young Cho

Dept. of Computer Engineering, Hankuk University of Foreign Studies,
Yongin Gyeonggi, Korea

Abstract. This paper propose a new rule match algorithm, called Rete-ECA, based on Rete for context-aware device control environment. The Rete-ECA exploits the natural Event-Condition-Action feature of device control situations. This enables the implemtation of the Rete-ECA algorithm to perform better than the Rete algorithm with smaller size and more flexibility. The Rete-ECA system is evaluated using a mouse-tracking environment. Rete-ECA consumes about 2% of the clock ticks consumed by original Rete.

1 Introduction

Context-aware devices enable users to interact with their environment in more meaningful ways. Although there is some debate about the precise meaning of context [1], context can be said to refer to information about a user's surroundings, location, or preferences that enables a context-aware application to provide better service to a user. Context-aware applications must acquire and update this information over time as it changes.

A context-aware application can use context information to connect to the devices around it. This requires that the application have some means of managing and controlling these devices. Rule-based systems provide designers a quick way to specify their application's behavior with respect to the devices the user wishes to use.

The focus of this paper is an inference engine that uses a new Rete-based algorithm called Rete-ECA for device control and management. The Rete-ECA inference engine uses device data to help determine the overall flow of control in a closed-loop system. This is achieved through two networks: one rule network dedicated to matching the patterns provided in a user-specified set of rules, the other dedicated to managing all devices in the network.

The Rete-ECA system is a Rete-based rule network for storing 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.

The overall goals of this paper are to demonstrate that a solely event-driven control system can be used to control a system's devices and that it can limit its own resource use by determining its own rule network structure.

Lee R. and Cho S..
Rete-ECA: A Rule-based System for Device Control.
DOI: 10.5220/0005140500950102
In *Proceedings of the International Workshop on Artificial Neural Networks and Intelligent Information Processing* (ANNIIP-2014), pages 95-102
ISBN: 978-989-758-041-3

In section 2, related work in rule matching algorithms and their use in context-aware environments is discussed. In section 3, the proposed rule engine system, Rete-ECA, is presented. In section 4, the mouse-tracking experimental environment is explained and demonstrates the power of separating the Event and Condition phases of ECA. Finally, this paper concludes with Section 5.

2 Previous Work

2.1 The Rete Match Algorithm

Rule-based systems [2] capture, 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, or in the case of Rete-ECA, a black box control system. The Rete match algorithm [3] is one of the most popular many pattern/many object pattern matching algorithms. It was initially designed for use as an interpreter for the OPS5 production system language. Productions are another term used to refer to if-then rules; hence, the Rete production system interpreter functions as a rule interpreter or inference engine. The Rete match algorithm can function as a pared-down rule-based system.

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. Objects are represented in the system by working memory elements. Multiple facts (the given object's attribute and value pairs) are associated with each working memory element. 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 [4]. The alpha network is used to perform simple tests of conditioin parts and the beta network is used for join the results of the alpha network.

Using these components, the rule interpreter executes in three phases: match, conflict resolution, and act. During the match phase, all the left-hand sides of the rules are evaluated. If any pattern matches are found, the interpreter enters the conflict resolution phase. In this phase, one matched rule is chosen, if any exist. Finally, in the act phase, the actions specified in the right-hand side of that rule are executed. A limitation of the Rete match algorithm is that the set of objects cannot change rapidly and the structure of the Rete network itself is static, which means adding, deleting or modifying a rule requires that the network should be rebuilt.

2.2 Other Notable Pattern-matching Systems

The Rete match algorithm stores element data at each alpha memory and each beta memory. The size of the beta memories can increase exponentially. One noteable system that addresses this issue is TREAT [5], a pattern matching algorithm that 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. Another noteable system is LEAPS

[6], an OPS5 rule set compiler that uses a lazy matching algorithm [7] to trade some completeness for much higher performance.

As a side note, an object-oriented version of the Rete match algorithm is implemented in Drools [8] and supplemented by added functionality described in Rete-OO [4]. This additional functionality enables Rete-OO to mimic the functions of neural networks, Bayesian networks, fuzzy logic systems, etc. closely, but at the expense of a more complex network and greater overhead.

Rete-Alpha Network Dual Hashing [9] is an optimization of the Rete match algorithm. It was developed as an inference engine to control composite context-aware services. Rete-ADH optimized the alpha network; Each alpha node contains a hash table that hashes variable names to a secondary hash table. This secondary hash table hashes a working memory element (called a fact) attribute to a list of related facts. Using the dual-hashing technique, Rete-ADH selects facts fast and reduces the size of the beta memories but suffers from inefficient condition checking for events.

MiRE [10] stands for **Mi**nimal **R**ule **E**ngine and was devised as a lightweight, context-aware rule-based system for resource-limited cellular phones. MiRE uses a modified Rete network for rule-matching. The Rete network contains only a fixed number of facts; when a new fact needs to be inserted and the Fact Manager already contains the maximum number of facts, a fact is selected and deleted from working memory according by oldest timestamp or a designer-defined principle. The new fact is then allowed to enter the system.

3 Rete-ECA for Device Control Environments

The main purposes of Rete-ECA is to provide a scalable platform for devices of varying sizes and functionalities in a dynamic control environment. This includes smaller mobile devices with higher constraints on the amount of memory and processing time that can be made available for the rule engine. To achieve the purposes, we design an event-driven Rete-based inference engine for device control and management. The inference engine avoids wasteful polling by executing only after an event has occured. It is portable enough to run on a variety of devices and fast enough to process all of the system's events within the time alloted. In addition, the rules can be adjusted dynamically.

3.1 The Overall Structure and Operation of Rete-ECA

The structure of the entire Rete-ECA system is shown in Figure 1. It consists of the inference engine, the rule parser, the rule memory, the device rule memory, the working memory, and the event processor. For the purposes of discussion, the phrase ”rule engine” will be used to refer to the entire Rete-ECA system, while ”rule network” will refer to the rule, rather than device, network used in Rete-ECA. ”Device network” refers to the Rete network used for matching sequences of device conditions provided in a rule's right-hand side.

The rule parser takes as input a plain text file of rules and parses them into the left-hand side and a portion of the right-hand side of a given rule. The left-hand side of a

rule specifies the sequence of conditions that must be true in order for the rule to be matched. The left-hand side is simply called a rule and stored in the rule memory. The portion of the right-hand side parsed from the text file specifies a sequence of conditions that must be true in order for the devices to be able to perform the rule's action. This portion of the right-hand side is known here as a device rule. The device rules are stored in the device rule memory.

Rules from the rule memory are used to build the rule interpreter. Device rules from the device memory are used to build the device rule interpreter. Both of these interpreters are Rete pattern-matching networks.

The working memory functions in Rete-ECA just as it does in the Rete match algorithm. It stores the system's relevant objects and their associated facts as working memory elements. These relevant objects are any objects whose facts may cause any rules' left-hand sides to be true. These objects include, but are not limited to, devices, users, and system state variables.

The event processor attaches event handlers to the objects in the system. If an object has been modified and this modification causes an event to be raised, the event processor forwards this object to the working memory. The working memory will then update the working memory element and the facts associated with this object, before passing the updated working memory element to the rule engine.

The system's execution cycle is split into three phases: Event, Condition, and Action.

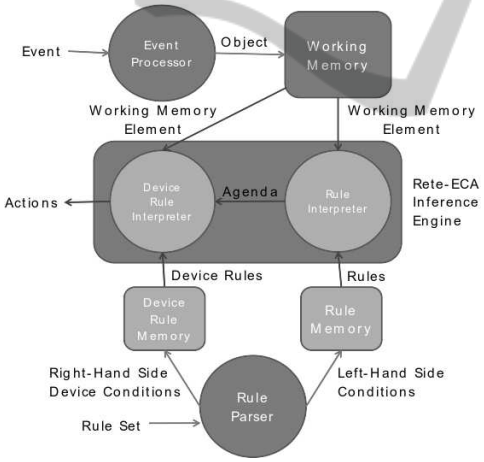

Fig. 1. Structure of the Rete-ECA system.

The operation of the Rete-ECA rule engine is entirely event-driven. The rule engine is triggered and run only after an event is known to have occured. Four types of events may occur.

The first type of event is timer-driven. Timer-driven events occur when the duration of a timer has elapsed. The second type of event is triggered by the user as it changes locations. These events can be said to be user-driven. The third type of event is triggered by changes made by the devices. Changes made to objects by devices require that the

objects have their information updated in the rule engine. The fourth and final type of event occurs when devices become available or unavailable to execute a rule.

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, and the left blue light is off, then if the left blue light is enabled, then turn on the left blue light and enable the feeder.

The rule, ”LeftLightOn” translates to

```
PSMouse:MOUSE(PastLocation == ”P”, CurrentLocation == ”S”)
LeftLight:BLUELIGHT(IsOn == False, DID == 0)
```


where MOUSE 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”. 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_ACT,” which is

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

This device rule matches the devices needed to execute the action. The left blue light and the feeder on the left side of the environment are enabled. If not all of the devices are matched, then, in this current implementation, the rule’s actions are not executed. A feature for backtracking through the device rule’s beta nodes would allow this to occur while incurring a small time penalty.

The final beta memory of each rule is connected to the rule itself. The rule is then connected to the terminal node. Each rule contains a reference to all of the beta nodes it uses during pattern matching.

Maintaining two networks has its own advantages and disadvantages. The most obvious disadvantage of creating two networks is the need to maintain both. Using two Rete networks incurs its own time and space overhead. There are, however, advantages to be had from maintaining the separate networks. The first is that some of the complexity of the rule network can be shifted to the device network. This results in shorter left-hand sides for rules and a corresponding decrease in the number of nodes and therefore size of memories in the rule network’s beta network, making left-hand side pattern matching faster.

Also, implementing the device network as a second Rete network allows for more

complex relations between devices to be represented and enforced through matching only devices whose conditions are compatible with each other. The device network allows for greater flexibility in determining which groups of devices are allowed to fire actions.

Another advantage is that the state of the device network at any given time provides information about how the rule network itself can be modified in order to speed the matching process.

3.2 Rule Network Modifications

There are two methods of changing the rule network: rule activation/deactivation and rule insertion/deletion.

Rules in the rule network switch between a status of ”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 matches. This prevents the matching process from entering the beta network through alpha nodes that are used for matching inactive rules.

Rule insertion/deletion operates differently. The network’s structure must be changed before the working memory element is removed and re-inserted. Rule insertion occurs in the same manner as it was when the rule network was first made, but a new algorithm must be implemented to support rule deletion. Past work on implementing and optimizing the Rete match algorithm has ignored or dismissed rule deletion as unnecessary, cumbersome, or too time-intensive to be of practical use. The authors instead opted to rebuild the entire network with the newly modified rule set.

It should be noted that, although the portions of the rule network that are used by the given rule are deleted, the rule itself moves to the rule network’s list of inactive rules. This is done to avoid any additional calls to the rule parser, which reads the rules from the text file in which they are written. If the rule is reinserted into the network at a later time, the rule is retrieved from this list of inactive rules and the portion of the network it needs is rebuilt.

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 2) used in these experiments is segmented into seven abstract locations called L, X, S, P, R, Y, and

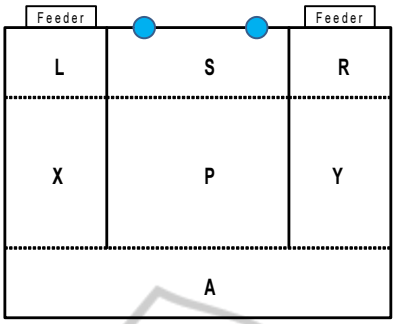

Fig. 2. Diagram of the bottom level of the experimental environment.

A. Transparent 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 of the left side and one on the right side.

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 occured. In the environment, the performance and size of several Rete-based matching systems are presented and compared with twelve rules.

At first, we measured the program sizes of the original Rete, Rete-ECA, Rete-ECA with rule activation/deactivation (Rete-ECA-AD), and Rete-ECA with rule insertion/deletion (Rete-ECA-ID), respectively. The size of the original Rete system is 876KB and all the variations of of the Rete-ECA system have similar program sizes, all of which are less than 700KB. This shows that the Rete-ECA system is suitable for small equipments.

Secondary, we compared the amount of virtual memory consumption of four systems. The values are 38.4MB, 37.2MB, 34.9MB, and 35.5MB for the original Rete, Rete-ECA, Rete-ECA-AD, and Rete-ECA-ID, respectively. Rete-based algorithms consume large memory to maintain rule networks and working memories. The dynamic feature makes the Rete-ECA system consume less memory than the Rete system.

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. Rete-ECA consumes the least clock ticks, approximately 2% of the clock ticks consumed by original Rete. It should be noted that all of the variations of Rete-ECA are event-driven. Among the variations, Static Rete-ECA consumes the least number of clock ticks (370,182). Rete-ECA-AD and Rete-ECA-ID consume about the same number of clock ticks, (404,210, 403,833, respectively). There is a trade-off between Rete-ECA and its variants according to the second and the third measurements. In performacne view-point, Rete-ECA is better than its variants but vice versa in memory usage view-point.

5 Conclusions

The focus of this paper is an inference engine that uses a new Rete-based algorithm called Rete-ECA for device control and management. The Rete-ECA inference engine uses device data to help determine the overall flow of control in a closed-loop system. The Rete-ECA exploits the natural Event-Condition-Action feature of device control situations. This enables the implemtation of the Rete-ECA algorithm to perform better than the Rete algorithm with smaller size and more flexibility. This system explicitly uses its events to determine when and which portions of the rule engine should activate at a specific time and seperates the Rete network into two networks: one rule network dedicated to matching the patterns provided in a user-specified set of rules, the other dedicated to managing all devices in the network. The experiments on mouse-tracking situations shows the Rete-ECA algorithm consumes only 2% of the original Rete algorithm.

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 partitioins and will have many benefis for parallel hardware architectures.

References

1. Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: Context Aware Computing for the Internet of Things: A Survey. IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, (2014) 414–454
2. Frederick, H.-R.: Rule-based systems. Communications of the ACM, Vol. 28, No. 9. (1985) 921–932
3. Forgy, C. L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, Vol. 19, No. 1. (1982) 17–37
4. Sottara, D., Mello, P., Proctor, M.: 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) 1535–1548
5. Miranker, D. P.: Treat: A Better Match Algorithm for AI Production Systems; long version: Tech. Rep. (1987)
6. Batory, D.: The leaps algorithm. Technical report, Austin, TX, USA,(1994)
7. Miranker, D. P., Brant, D. A., Lofaso, B. J., Gadbois, D.: On the Performance of Lazy Matching in Production Systems. AAAI, (1990) 685–692
8. Proctor, M., Neale, M., Lin, P., Frandsen, M.: Drools (2014)
9. Kim, M., Lee, K., Kim, Y., Kim, T., Lee, Y., Cho, S., Lee, C.-G.: Rete-adh: An improvement to rete for composite context-aware service. Int. Journal of Distributed Sensor Networks, (2014)
10. Choi, C., Park, I., Hyun, S. J., Lee, D., Sim, D. H.: Mire: A minimal rule engine for context-aware mobile devices. 3rd Int. Conf. on Digital Information Management, IEEE, (2008) 172–177