IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 11, November 2015.
www.ijiset.com
ISSN 2348 – 7968

Dynamic Rule Modification Method for a Light-weight Rule Engine

Seung-Kyu Yoo[1] and Sang-Young Cho[1]

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

Abstract
Context-aware services provide users with best-fitted services according to the current situation contexts of users. Usually rule-based systems are used as base engines for context-aware services due to their clear representation of situation contexts and corresponding actions to be served. A rule set used for a rule-based system should be dynamic if environment where context-aware services are applied is changing over times. Thus, a dynamic rule modification feature for a rule-based system is necessary. This paper describes a design and implementation of a dynamic rule modification method for a Rete-based rule inference engine called LwRE. LwRE is developed to minimize memory consumption for resource-constrained smart things of IoT. The implemented modification method is verified using a rule set of a mouse training system. LwRE can be used to provide context-aware services in dynamically varying environments.

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

1. Introduction

Smart things of the Internet of Things (IoT) environment usually have various sensors, gather and filter sensor data to get valid data, and extract context information for intelligent interaction with their environments [1]. Therefore, smart things are able to extract useful situation information from massive raw data of IoT environment and to provide context-aware services. In this paper, context refers to information about devices available to the user at his/her location, device information, including users' location or preferences. Context information may be continuously changed or updated depending on the change of the information of a user and devices. Context information are changed, for example, when a user moved to a different location or is doing special actions, a sensor's measured value is changing due the change of physical environment state, or outputs of devices are varying. A context-aware service application can provide users with best-fitted services according to the current situation contexts of users and environment by acquiring situation data, recognizing context processing the data, recognizing context, and providing the best-fitted service [2].

As context-aware services require knowledge modeling and reasoning process for recognizing context, various reasoning methods such as rule-based inference, supervised learning, unsupervised learning, fuzzy logic, ontology, and probabilistic inference are necessary. Among the methods, rule-based inference method has been most used in various application areas due to its clear representation of situation contexts and corresponding actions to be served [3]. Rule-based systems use rules of the IF-THEN structure that specifies the THEN action part of a rule is performed when the IF condition part of the rule is satisfied. The inference engine of a rule-based system continuously evaluates its rule set and finds the satisfying rules, and executes the action parts of the rules [4]. In context-aware services, rules are used to specify the actions of the service providing devices when situation data composing a specific context is changing. Then, rule engines find the rules satisfied with the input sensing data and perform the corresponding actions. The action can be an operation of devices or providing service information.

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. These changing circumstances enforce a set of rules to describe an environment situation to be changed over times. This paper describes a design and implementation of dynamic rule modification methods for a Rete-based rule inference engine called LwRE [5].

LwRE (Light-weight Rule Engine) is based on the Rete algorithm that is one of the most commonly used one for rule inference engines of rule-based systems [6]. Though the Rete algorithm is a general-purpose algorithm that can be used in various applications, it is difficult to use the algorithm for a resource-constrained system because it consumes a lot of memory to speed up the rule matching process. To overcome its memory and performance drawbacks, there have been several studies on algorithm improvement and, recently, studies are focusing to apply the algorithm to context-aware services [7-11].

LwRE is designed to provide context-aware service in a system with low computational and memory capacity. The engine adopted the Event-Condition-Action (ECA) model. The ECA model confirms current Condition when Event occurred, and if the condition evaluates to be true, then the action is executed. This model enables the system to react to event and carry out actions only if the condition was turned out to be true, thereby avoiding unnecessary rule checking [12-13]. The implemented rule modification method is verified using a rule set of a mouse training system (MTS) and enable LwRE be used in dynamically varying environments.

This paper is organized as follows: section 2 discusses Rete algorithm and previous work. In section 3, we explain the structure and operations of LwRE and suggest a dynamic rule modification method for LwRE. The verification of the method is discussed in section 4. Finally, this paper concludes with section 5.

2. Related Work

2.1 Rete algorithm

The Rete algorithm [5] is one of the most popular rule 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 rule network. The structure of this network is determined by the condition part of all the rules that the system uses. The rule network is a directed acyclic graph of nodes and is divided into the alpha network and the beta network. 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
      l : Lightning( Voltage  >  1GV )
      m : Computer( address  = l.location )
   then
      sendAlarmTo( m.owner )
   end
```

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 "NULL".

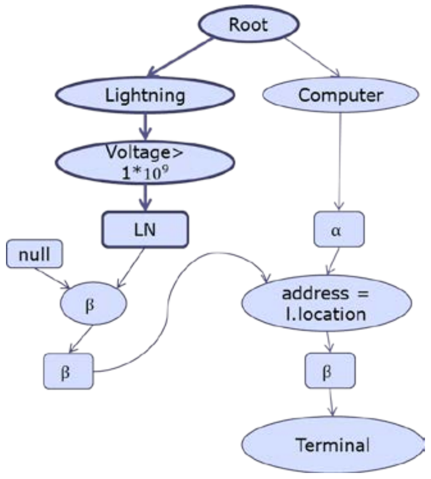

Fig. 1 Rete rule network example

There are two type nodes, one that checks for the object type Lightning and the other that checks for the object type Computer. The alpha condition, "Voltage > 1GV", 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 Lightning'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.

Though Figure 1 shows the rule network with only one rule, rule networks can be very complex because the number of rules is over hundreds or thousands in real application areas. The Rete match algorithm stores element data at each alpha memory and each beta memory. The size of the beta memories can increase exponentially. Therefore, there have been several researches to enhance performance and to reduce memory consumption for the Rete algorithm.

2.2 Improvement of Rete algorithm

In case of rule sets which have complex relationships between objects, the closer beta memory to a terminal node consumes more memory. Therefore TREAT [7] removed the memory requirement in the beta network, but at the cost of increased processing time and a more complex memory in the alpha network. LEAPS [8] removed beta memory and accelerated the computational speed through selecting only one rule that satisfy current situation using lazy evaluation with stack structure, but it incurs inaccurate matching results occasionally. Rete-OO [9] is an object-oriented version of Rete algorithm, with features that simulate neural network, Bayesian network, and fuzzy logic but computational burden involved due to complex network structure.

Rete-ADH (Alpha Network Dual Hash) is an inference engine to control integrated context-aware services of a SMART office, using double hashing in alpha network of the Rete algorithm [10]. One of two hash tables stores fact variables satisfying rule conditions and other stores all the facts satisfying rule conditions. Rete-ADH reduced memory size and speeded up the matching process by choosing the fact with the highest probability in beta checking. But, facts that satisfy the condition are overlapped in alpha network and there are extra burdens in operating double hashing. MiRE [11] is a lightweight rule engine to offer rule-based context-aware services in a resource-constraint mobile phone. It is based on Rete algorithm and limits the number of facts to reduce memory consumption. It also manages unchanged and varying facts separately. This method, consequently, helps control memory usage suitable for context-aware service of a mobile phone.

MidSen[12], developed as a middleware supporting various context-aware services of sensor network, does not use Rete algorithm but adopts rule-based architectures. It abides by the ECA model that detects events explicitly, checks conditions of the related rule, and performs actions of the rule that satisfies the conditions. Because MidSen requires to record fact information describing context and to re-evaluate rule sets at every event, it is not efficient in terms of time. [13] uses Ubiquitous chips which implement ECA engine for rule-based control of I/O devices. The Ubiquitous chip can be linked with I/O devices, embedded with applicative ECA rule sets and used to provide context-aware service by connecting chips in parallel. As it supports a limited number of connections to sensors, the cost of hardware may be increased depending on the number of sensors.

3. Dynamic Rule Modification for LwRE

The main purpose of LwRE is to provide a platform for smaller devices with higher constraints on the amount of memory in a dynamic control environment. To achieve the purpose, the inference engine eliminates beta memory and manages hash tables to avoid performance degradation due to memory elimination. LwRE changed the structure and operations of alpha memory so as to eliminate redundancy of alpha nodes by maintaining conditions at the minimum level. 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 LwRE

A rule structure is composed of five elements that are rule itself, condition, attribute, join, and action. Each element is stored in the corresponding hash table. Each of elements has its name that is used to key to access each other. The rule hash table and condition hash table have storage for object keys that satisfy rule or condition to serve as beta memory and alpha memory of the Rete network. Conditions of a rule represent fact-checking for a specific type and consist of attribute conditions and join conditions. Attribute conditions represent comparison of the attribute value of an object with some constant value and is implemented into attribute nodes of the rule network. Join conditions represent inter-object comparison between objects of different types. Figure 2 shows the rule network structure of LwRE.

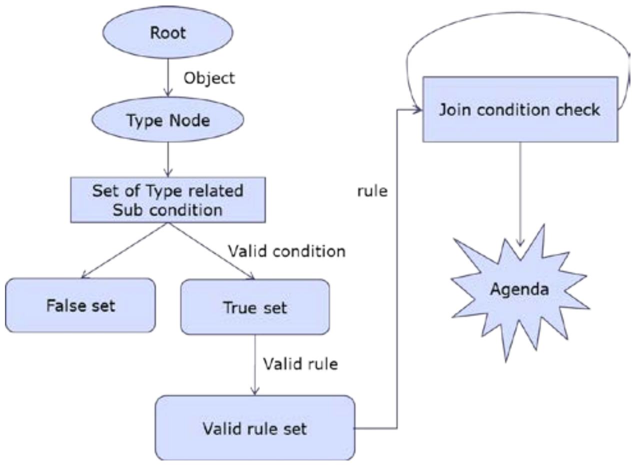

Fig. 2 The rule network structure of LwRE

Root node is connected to type nodes. Each type node is connected to attribute nodes that should be checked when an attribute value of a device of the type is changed. After checking attribute conditions, the valid condition’s key is stored in True set and the invalid key is stored in False set.

Valid rule set stores rules that are partially satisfied with attribute conditions in True set. For valid rules in Valid rule set are checked whether their join conditions are satisfied. Finally the rules that satisfy all the conditions fire their actions.

To explain attribute checking process, we assume the following three conditions and one input.

Cond0: (Type='Bluelight', DID==2 && State=="ON")
Cond1: (Type='Bluelight', State=="ON" &&
Enable="TRUE")
Cond2: (Type='Bluelight', Enable=="TRUE" &&
DID==2)
Input: (Type='Bluelight', DID==1, Enable="TRUE",
State="ON")

Condition 0 is satisfied if DID of a blue light is 2 and the current state is ON. Condition 1 is satisfied if a blue light is ON state and currently enabled. Condition 2 is satisfied if DID of a blue light is 1 and it is now operating. Figure 3 shows the corresponding rule network for the above conditions and the attribute checking process.

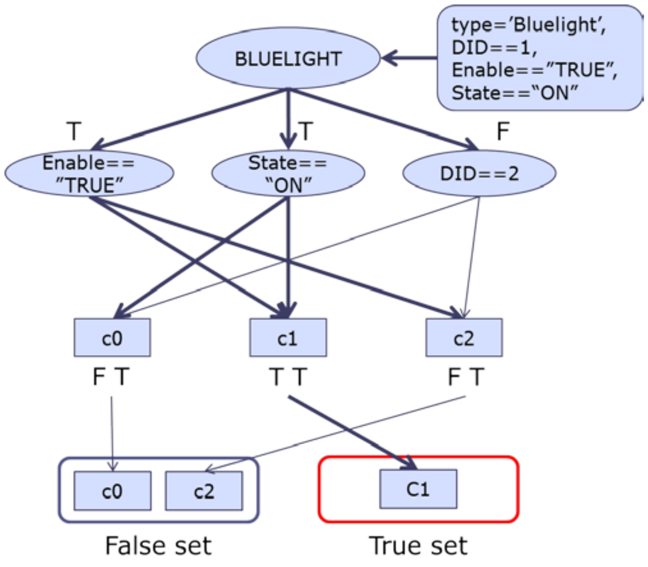

Fig. 3 A rule network of LwRE and attribute condition process

There are three attribute conditions and two of them are true for the given the input facts that a blue light's DID is 1, is enabled, and is turned on. Using the result of attribute condition checking, three conditions are checked whether all their attribute conditions are satisfied with the input. Only Condition 1 is satisfied and the key of Condition 1 enters True set and others enter False set.

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 two rule network modification methods: one is rule activation/deactivation and another is rule insertion/deletion method. Two methods can be used to discard unnecessary rules in rule matching processes and it makes the LwRE algorithm perform better than the algorithm without modification. The first method is simple but it uses more memory than the second method because the first method maintains all the rules involved in memory.

To implement the rule activation/deactivation method, each rule has a flag that indicates whether the rule is active or inactive. Figure 4 shows the activation/deactivation operation with the active flag of a rule.

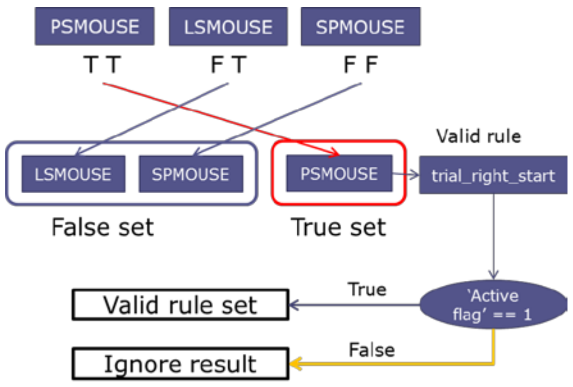

Fig. 4 Rule activation/deactivation method

A rule that satisfies its conditions goes into Valid rule set to check join conditions. In activation/deactivation method, the active flag is checked and the rule is ignored to enter Valid rule set if the flag is set to inactive. This method is also used when the system is initialized or rules are inserted to construct a rule network to avoid unexpected behaviors of the system.

LwRE can insert/delete rules into/from a rule network during operation. The rule insertion is performed by inserting the corresponding inactive rule into the existing rule network as like the initialization step. The rule deletion needs two operations: one is deletion from rule memory and the other is deletion from rule network. The deletion from rule memory performs the following steps.

1. For the conditions of the rule, remove the rule key from the parent rules of the conditions.
2. Remove conditions that have no parent any more.

3. Remove the keys of the conditions that have to be removed from the parent conditions.
4. Remove the attribute and join conditions that have no parent any more.
5. Remove conditions that have no child attribute and join conditions.
6. Remove rules that have no child conditions.

Rule deletion is the repeated operations of deleting the rule key from its children and deleting children that have no parent. If an attribute condition is removed from rule memory, the key of the condition should be deleted form rule network. The deletion of an attribute condition from rule network is performed when the attribute condition is referred to without searching the attribute condition in rule network.

4. Experiments

The LwRE system described in the previous section was verified with a mouse training system (MTS). Mice are placed in this system and their behavior is observed and studied by psychology researchers. If the mice behave in a particular way in relation to the system, they can be rewarded or punished. The rewards and punishments are determined by the researchers in a way that meets their research objectives. The system (see Figure 5) used in verification is segmented into seven abstract locations called L, X, S, P, R, Y, and 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 on the left side and one on the right side.

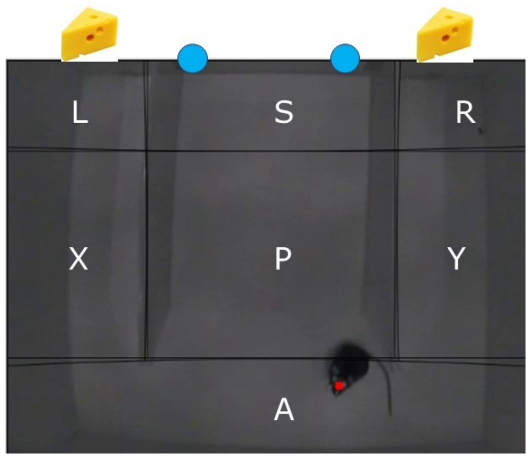

Fig. 5 Diagram of the bottom level of the experimental environment.

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 and LwRE compared with twelve rules.

At first, we measured the program sizes of the two systems. For the 12 rules used, LwRE generates 16 attribute nodes and Rete generates 20 alpha nodes. Though the total number of attribute nodes is 45, LwRE eliminates the redundant nodes successfully and is superior to Rete algorithm that eliminates redundancy at the condition level. The size of the original Rete system with one rule is about 10 MiB that is similar to LwRE. But, the size gap between LwRE and Rete is increasing as the number of rules is increasing. The engine size of Rete is 26 MiB at nine rules but that of LwRE is 10.5 MiB. This means that LwRE engine uses almost the same size of memory regardless of the number of rules.

Secondary, we compared the amount of dynamic memory consumption of two systems. The values are 12 KiB and 133 KiB for the LwRE and Rete, respectively. Rete-based algorithms consume large memory to maintain rule networks and working memories. The run-time memory consumptions of two systems have little variance but the sizes of dynamic memory consumption have big gap. The LwRE system shows its optimized memory consumption and thus is appropriate for the resource-constraint system.

With the rule modification functions of LwRE, the memory consumption can be controlled more precisely. If the memory requirement is strict, then the rule inserting/deletion method of LwRE is better method than the activation/deactivation method.

5. Conclusions

The focus of this paper is a design and implementation of a rule modification method for the LwRE inference engine. 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. LwRE addressed the memory issue and optimized memory consumption by removing beta memory and maintaining hash tables at the attribute condition level. This paper proposed dynamic rule modification method that enables the LwRE to be used in a dynamically varying environment. The experiment shows that the rule modification methods are working well and contributing to the memory consumption issue.

In the future work, we will apply the LwRE engine to extended problems in not only smart things of IoT but also smart building or home automation control systems.

Acknowledgments

This work was supported by Hankuk University of Foreign Studies Research Fund of 2015.

References

[1] Internet of Things[Internet], https://en.wikipedia.org/wiki/Internet_of_Things.
[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context Aware Computing for the Internet of Things: A Survey”, IEEE Communications Surveys & Tutorials, Vol. 16, No. 1, 2014, pp. 414-454.
[3] B. Y. Lim and A. K. Dey, “Toolkit to support intelligibility in context-aware applications,” in Proc. of 12th ACM international conference on Ubiquitous computing, New York, USA, 2010, pp. 13-22.
[4] F. Hayes-Roth, “Rule-based systems”, Communications of the ACM, Vol. 28, No. 9, 1985, pp. 921-932.
[5] S.-K. Yoo, “LwRE: Light-weight rule engine for context-aware service,”, Master’s thesis, Hankuk University of Foreign Studies, Korea, p. 58, 2015.
[6] 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.
[7] D. P. Miranker, “Treat: A Better Match Algorithm for AI Production Systems; long version”, Technical report, University of Texas at Austin, 1987.
[8] D. Batory, “The leaps algorithm”, Technical report, University of Texas at Austin, 1994.
[9] 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.
[10] 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.
[11] 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.
[12] P. Patel, S. Jardosh, S. Chaudhary, and P. Ranjan, “Context aware middleware architecture for wireless sensor network,” in Proc. of IEEE International Conference of Services Computing, 2009, pp. 532-535.
[13] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio, “Ubiquitous chip: A rule-based i/o control device for ubiquitous computing,” in Pervasive Computing, Lecture Notes in Computer Science, vol. 3001, Springer Berlin Heidelberg, 2004, pp. 238-253.

Seung-Kyu Yoo received his Bachelor’s Degree in Computer Engineering from Hankuk University of Foreign Studies, Korea, in 2013, and M.S. degree in Computer Engineering from Hankuk University of Foreign Studies, Korea, in 2015. His research is currently associated with embedded system, IoT solution, and cloud service..

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.