Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 507160, 11 pages
http://dx.doi.org/10.1155/2014/507160

Research Article

RETE-ADH: An Improvement to RETE for Composite Context-Aware Service

Milhan Kim, Kiseong Lee, Youngmin Kim, Taejin Kim, Yunseong Lee, Sungrae Cho, and Chan-Gun Lee

Department of Computer Science and Engineering, Chung-Ang University, Seoul 156-756, Republic of Korea

Correspondence should be addressed to Chan-Gun Lee; cglee@cau.ac.kr

Received 4 December 2013; Accepted 1 April 2014; Published 24 April 2014

Academic Editor: Jongsung Kim

We propose a new pattern matching algorithm for composite context-aware services. The new algorithm, RETE-ADH, extends RETE to enhance systems that are based on the composite context-aware service architecture. RETE-ADH increases the speed of matching by searching only a subset of the rules that can be matched. In addition, RETE-ADH is scalable and suitable for parallelization. We describe the design of the proposed algorithm and present experimental results from a simulated smart office environment to compare the proposed algorithm with other pattern matching algorithms, showing that the proposed algorithm outperforms original RETE by 85%.

1. Introduction

The past decade has seen the dropping costs for wireless personal devices such as smart phones and tablets while the capabilities of those devices continue to evolve rapidly. Interconnecting these computing devices or distributed sensors can be used to provide intelligent services, using a variety of sensing technologies of various sizes, from simple motion sensors to electronic tags or video cameras [1]. Furthermore, recent advances in software technology and computing devices have enabled revolutionary and user-customized digital services [2] such as ubiquitous computing. These differ from conventional communication models, requiring sophisticated integration of huge amounts of information on the fly to enable performing appropriate actions in a timely manner.

Because of these complex aspects, composite context-aware (CCA) architecture, which we describe in detail in Section 2, is promising for ubiquitous computing. Context-aware techniques can be used to provide the user with useful and intelligent applications by taking into account contextual information from the user's environment. The CCA architecture is composed of several elements, which include an *inference control function, interpretation function,* composite context information (CCI) *repository, and inference engine*. Among them, the *inference engine* which uses the *event-condition-action (ECA) engine,* is the key in speeding up CCA services. To construct an efficient *inference engine*, one of the most reasonable solutions is to implement an *ECA engine* by using pattern matching algorithm developed for production systems.

The RETE algorithm [3] is one of the most efficient forward inference algorithms that can be used for constructing an ECA engine; however, it has a few drawbacks [4–6]. While there have been a number of contributions to improving the pattern matching algorithm [7–10] in recent decades, not many of them considered an ECA engine within the CCA architecture as the target environment. In addition, although past contributions have led to notable performance improvements, the improved algorithms have also been more complex. For example, although the RETE' algorithm [10] has shown a performance improvement of more than 80% relative to RETE, it requires a complicated algorithmic structure and introduces time stamp in its data structure.

In this paper, we propose a new pattern matching algorithm called RETE-Alpha network Dual Hashing (RETE-ADH). We developed RETE-ADH for CCA services, and herein we describe its validation through simulation in a CCA environment. It is a relatively simple algorithm that maximizes the use of referencing rather than comparing or searching all nodes. This characteristic makes RETE-ADH faster than other existing algorithms, as we show in the evaluations in Section 4. Because of its concise network structure, RETE-ADH is scalable, and thus it can handle the varying characteristics of recent complex network services. Furthermore, we predict that RETE-ADH can be implemented in parallel hardware for the same reason. The proposed algorithm efficiently searches only rules that can be fired by reconstructing the alpha network with hash tables. This can increase the speed of pattern matching considerably while providing a full set of matched results like RETE.

2. Background and Related Work

2.1. Composite Context-Aware Service Architecture. Context can be considered as a set of information that includes a user's activity, location, personal preferences, and current status. The most widely accepted formal definition of context is that of Abowd et al. [11]: "Context is any information that can be used to characterize the situation of an entity. An entity can be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." In a previous study [12], we have classified context into *unitary* and *composite* contexts. We have defined the *unitary* context as a basic building block that is not further divisible. Also, as shown in Figure 1, we have defined the composite context information as a high-level context abstraction by integrating or composing *unitary* context information with related multientities [12]. In other words, to provide composite contextual service, multiple *unitary* contexts can be combined. For example, suppose that a tourist is planning to visit an attraction near his home. Then, *visiting an attraction near home* would be considered as the composite context. To construct the composite context, the following various *unitary* contexts can be related to the composite context: *weather, allowed time, distance to the site, budget, accompanying people, transportations,* and so forth. Future context-aware services are expected to manipulate such complex composite context.

Figure 2 shows our proposed CCA service architecture. When a user sends composite context information to the CCA service, the service architecture requests/subscribes the CCI through the service layer. The user requests the CCI from the context-aware service, and then the service responds to the user's request from the CCI interface control function if the service is immediately able to find the proper presence information. Otherwise, the user subscribes the CCI to the context-aware service. Then, the composite context-aware service processes the CCI through the underlying CCI interface control function. Once the CCI interface control function finds the proper presence information for the context, the information is sent to the user. After receiving this request/subscribe command, the CCI interface control function saves the CCI to the CCI repository. The CCI repository can be utilized for later requests/subscriptions. In the CCI interpretation function, the request-subscribe command is differentiated through a function-type decision process and passed to the request/response and subscribe/notify processes. After the request/subscribe command is processed, the CCI is extracted by the CCI extraction function and then the corresponding rules are parsed and translated. By using the rule pattern matching algorithm, the inference engine finds proper presence information for the context. We adopt our proposed pattern matching algorithm for this inference engine.

Figure 3 shows an example of the processing of CCI in the proposed composite context-aware service architecture. The context-aware service should increase the capabilities of the user's smart devices in various contexts, such as the home, office, or shopping mall. Composite context-aware applications can monitor such regions and modify their behavior accordingly to help provide comfortable lifestyles. Based on this, the CCI needs to be expanded to consider the user's current environment. For example, the CCI can be expanded by using device ID, user ID, service ID, and location ID. The device can have unique parameters such as device name, type, provider, supported services, connected network status, location, and owner. In a similar fashion, the user ID has unique parameters such as user name, device name, user location, and user's status.

2.2. ECA Architecture and Pattern Matching Algorithms. The ECA pattern is composed of three modules: event, condition, and action. Each rule is expressed as IF <event-condition> THEN <action>. The <event-condition> part of the rule specifies the situation under which the actions are enabled, and it is composed of a logical combination of events. An event models some occurrence of interest in an application or in an environment. The <action> part of the rule is composed of one or more actions that are triggered whenever the <condition> part is satisfied [1].

The ECA architecture can be used to support the composite context-aware service, and one of the most efficient ways to construct an ECA engine is to adopt a pattern matching algorithm. Of course, we can adopt existing rule based pattern matching algorithms; for this reason, we analyze existing algorithms for the inference engine such as RETE [3], TREAT [5], and LEAPS [7]. However, using an algorithm specifically developed for CCA services would be expected to improve the quality of service; accordingly, herein we propose a new pattern matching algorithm.

2.3. Pattern Matching Algorithms. Rule-based systems execute actions based on the rules that are fired by the incoming facts. Each fact is an expression of a certain situation or environment in the real world. Each rule is a predetermined method for how the system should behave in a certain situation.

A typical rule-based system is composed of working memory, a knowledge base, an inference engine, and an action performer. The working memory is a space in which facts are saved; facts are frequently updated and changed in the system. The knowledge base is a space in which rules

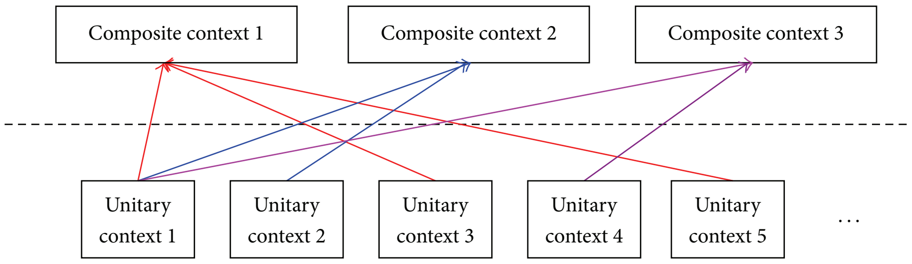


FIGURE 1: Derivation of composite context.

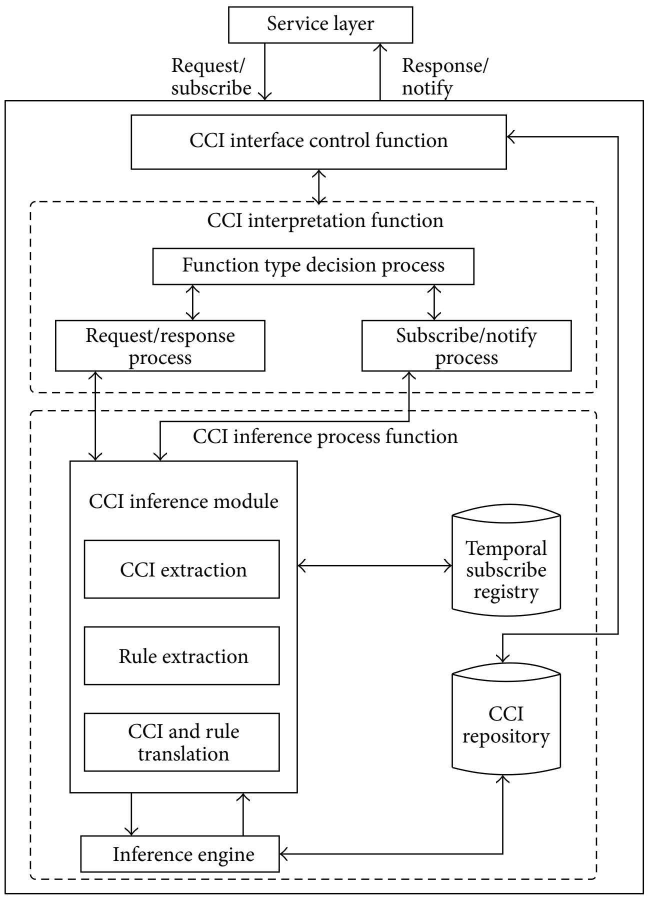


FIGURE 2: Architecture of composite context-aware service.

are saved. Conditions are tested under the criteria of the rules. The inference engine is a core part of the rule-based system; it checks whether the facts satisfy the rules based on a substitution method during each cycle of the inference engine. This test should be repeated for all the rules; therefore, there are usually a very large number of calculations in each cycle of the inference engine. This process in the calculation is called *pattern matching*. After finally deciding on a rule, the <action> part is executed; this process is called "firing." When a rule is fired, the facts in the working memory can be updated. The pattern matching time represents almost all of the processing time; for this reason, reducing the pattern matching time would directly improve the system's overall performance. An inference engine that uses the RETE or TREAT algorithm has a space called an agenda. The agenda temporally saves rules whose <condition> part matches the facts; this set of rules is called a *conflict* set.

In this section, we will give a brief overview of RETE, TREAT, and LEAPS, which are the most popular pattern matching algorithms that are used in many inference engines. We will analyze these algorithms and discuss their limitations.

2.3.1. RETE. Most pattern matching algorithms save in their working memory any partial matches produced during the previous inference cycle. Thus, they can avoid reevaluating the entire set of facts whenever changes are made. The RETE algorithm [3] is one such pattern matching algorithm; it maintains a network of partial matches to improve run-time efficiency. This network is composed of nodes that contain the facts. Each fact can be mapped to a token, which consists of a tag and a list of data elements. The tag indicates that the corresponding token has been added to or deleted from working memory.

There have been many previous efforts to address RETE's problems [4–6]. Among them, we specifically consider the issue of beta memory explosion. The RETE network has 2-input nodes, which are referred to as beta nodes. They send their outputs to beta memory, and because the RETE retains partial matches for performance reasons, the number of beta nodes increases rapidly, and thus maintaining the beta memory consumes vast memory resources.

2.3.2. TREAT. One of the main goals of the TREAT algorithm is to overcome a major problem of the RETE, namely, to reduce the overhead of network management. This algorithm is motivated by McDermott's hypothesis: "It seems highly likely that for many production systems, the retesting cost will be less than the cost of maintaining the network of sufficient tests" [13]. Unlike RETE, TREAT does not use beta memory; thus, it significantly reduces the overhead of network management. As a result, TREAT exhibits a maximum 50% better performance than RETE [5].

To avoid the use of the beta memory, TREAT recomputes the matches repeatedly [14]. This means that the job of finding the alpha nodes that correspond to the input of beta nodes is done repeatedly, which may render TREAT inappropriate for bounded algorithms.

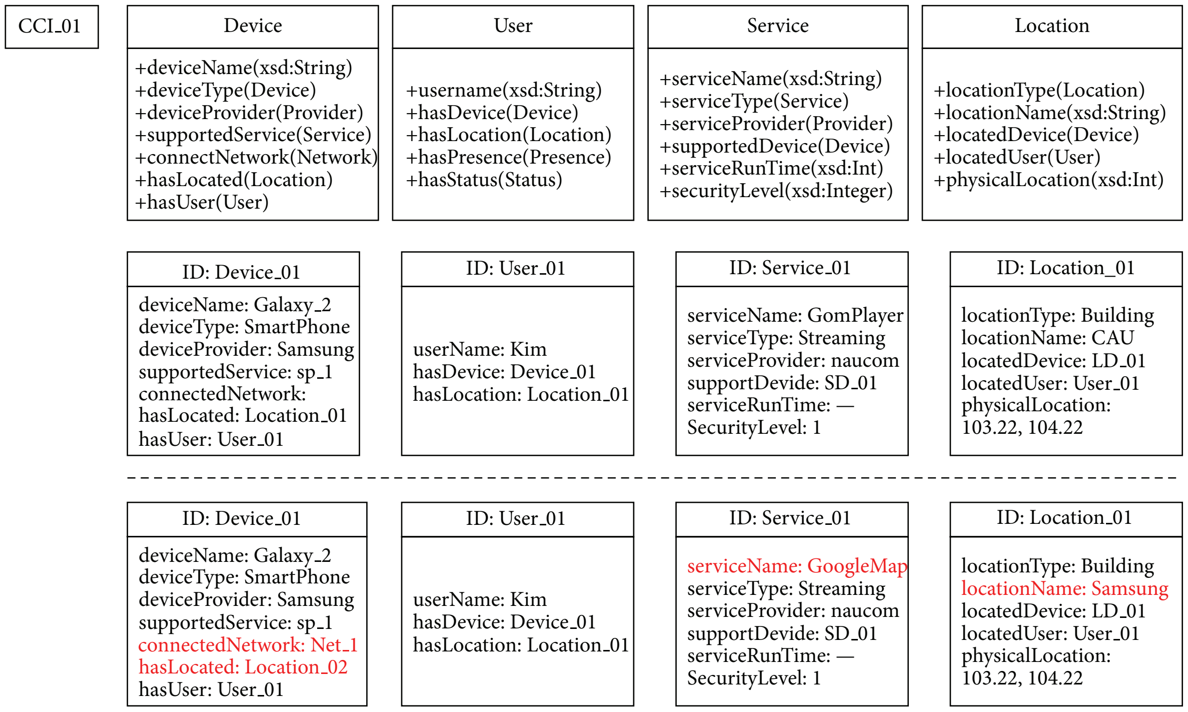


FIGURE 3: Example of composite context information structure.

2.3.3. LEAPS. Unlike RETE and TREAT, which use eager evaluation techniques, LEAPS uses lazy evaluation. Instead of generating all possible matches, LEAPS computes at most one match per cycle. It discards the conflict set, using a stack structure instead. Thus, it can reduce the computing time relative to the case of RETE and TREAT, which require processes for conflict resolution. Furthermore, LEAPS does not need beta memory. The stack management cost for LEAPS is known to be very low compared to most applications. LEAPS shows better performance than other algorithms, especially when the conditions are complex [15].

In spite of the advantages of LEAPS, there are some obstacles to use it generally. Its characteristic of producing at most one match per cycle makes it unsuitable for some applications [16]. Such applications include the simulator used herein to evaluate the proposed algorithm, and also general ECA engines. Similarly, composite context-aware services require a full set of match instances.

3. Proposed Algorithm

3.1. Overview. As mentioned before, RETE may consume a lot of resources due to its heavy use of beta memory. TREAT requires less memory than RETE, but may undergo an explosive amount of recomputations. LEAPS is more efficient than RETE or TREAT in terms of time and storage, but it may not fit some applications, including our own scenario in which its output of at most one match is not enough. For this reason, we propose a new pattern matching algorithm called RETE-ADH which extends the RETE algorithm with hashing techniques.

One of the primary features of the proposed algorithm is its adoption of double hashing in the alpha network; this increases the matching speed. Double hashing also reduces the number of beta nodes, thereby reducing the volume of the conflict set. This change, consequently, transforms many comparison operations into simple referencing operations; this is the main contributor to performance improvement by this algorithm.

Recently, a few approaches including alpha network hashing have been reported [8, 9]. In the alpha network, the alpha nodes are used to evaluate literal conditions of the facts. The fact data propagates through the next alpha node when it satisfies the current literal condition. The alpha node hashing is effective in the process when the propagation goes from an object-type node to an alpha node [8, 9]. In these approaches, an alpha node is added to a type-node, and the literal value is added as a key to the alpha node.

3.2. Core Algorithm. In our proposed algorithm, we use double hashing as follows.

(i) Each alpha node is hashed to variable nodes.

(ii) Each variable node consists of a variable name and a secondary hash table.

(iii) Each entry in the secondary hash table consists of a pair of fact attributes and a list of the related facts.

Note that in previous approaches using alpha network hashing [8, 9], all the facts in the alpha network have to be searched to build the beta network. In contrast, the proposed algorithm avoids useless alpha nodes by using the secondary hashing table. For this reason, RETE-ADH can reduce the volume of the beta network and turn some comparison operations into referencing operations.

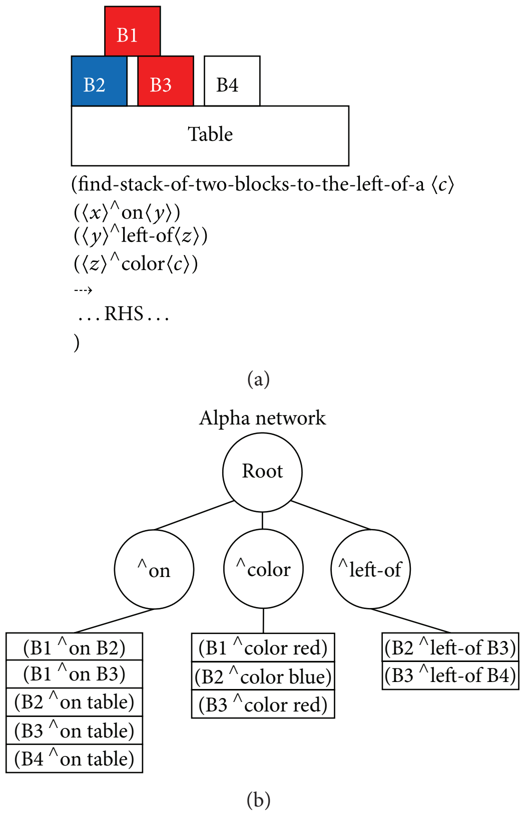
3.3. Case Study. Assume that we have the set of rules shown in Figure 4(a). In Figure 4, the rule searches for a stack of two blocks to the left of a block with a specific color. This rule has three conditions, which are enclosed by parentheses. Within each condition, let us denote a variable by enclosing it with angle brackets. For example, $\langle x\rangle$ indicates a variable x in the condition. Constants and identifiers are not enclosed by brackets. To fire this rule, we need facts satisfying the three conditions, which we will refer to as c_1, c_2, and c_3.

Assume that we have the fact (b1 ^ on b2), which satisfies condition c_1. Then, the fact that satisfies condition c_2 will be (b2 ^ left-of b3), since b2 in condition c_1 and b2 in condition c_2 should be matched. The fact satisfying condition c_3 will be (b3 ^ color $\langle c\rangle$). The $\langle c\rangle$ in the condition specifies the color that the user wants. Similarly, we can list the matched conditions as shown in Table 1. The RETE pattern matching algorithm constructs the alpha network with Table 1 as shown in Figure 4(b).

Figure 5 shows the alpha network of RETE-ADH based on the rule example shown in Figure 4(a). The matching process of RETE-ADH is similar to that of RETE; the difference is in how to construct the alpha network. In the RETE algorithm, one node is chosen in the alpha network and is attempted to be matched with the nodes of the beta network by using all the facts. In contrast, the RETE-ADH algorithm chooses facts in the alpha network that are highly likely to match with beta nodes. Assume that we try to match c_1 with c_2. In this case, c_1 has two variables $\langle x\rangle$ and $\langle z\rangle$. Because the hashing table is composed of the variables, RETE-ADH tries searching with these variables. Since c_2 has no $\langle x\rangle$, RETE-ADH searches c_2 with $\langle y\rangle$ of c_1. The c_2 has an identifier ^ left-of, and c_2 has a hashing table that sets $\langle y\rangle$ and $\langle z\rangle$ as a key. Each entry in hashing table of c_2 consists of a pair of fact attributes and a list of the related facts. In Figure 5, B2 and B3 are substituted into the $\langle y\rangle$ of c_1. Therefore, we can find the fact (B2 ^ color blue) by using the primary hashing table that sets $\langle y\rangle$ of c_2 as a key and the secondary hashing table that has B2 as a key. In this manner, we can find (B3 ^ color red) with B3. Note that RETE-ADH searches facts using the double hashing table instead of searching all of the facts of the alpha node, as mentioned above.

In the previous example, RETE has to apply 24 combinations of the facts to find condition matches in Figure 4(b); contrastingly, RETE-ADH tries only 4 combinations. In this specific example, the number of beta nodes is the same for both RETE and RETE-ADH; if there are many conditions and facts, there will be a huge number of beta nodes generated in RETE, making the matching execution time of RETE even worse than in Figure 4 example.

3.4. Characteristics. In our study, among RETE, TREAT, and LEAPS, we chose to extend RETE because it best fits our application purpose. If we were to use TREAT for our application, the recursive matching calculation of TREAT could become explosive because there is a huge number of facts in the composite context environment. We also consider LEAPS to be unsuitable because it produces at most one match per cycle, meaning that we cannot choose the most appropriate service by using LEAPS alone.



```
(find-stack-of-two-blocks-to-the-left-of-a ⟨c⟩
(⟨x⟩^on⟨y⟩)
(⟨y⟩^left-of⟨z⟩)
(⟨z⟩^color⟨c⟩)
-->
...RHS...
)
```

Figure 4: Rule example and its alpha network in RETE.

Table 1: Facts of the ω_n, where $n \in \{1, 2, \ldots, 9\}$.

ω_1:	(B1 ^on B2)
ω_2:	(B1 ^on B3)
ω_3:	(B1 ^color red)
ω_4:	(B2 ^on table)
ω_5:	(B2 ^left-of B3)
ω_6:	(B2 ^color blue)
ω_7:	(B3 ^left-of B4)
ω_8:	(B3 ^on table)
ω_9:	(B3 ^color red)

Figure 6 shows a concise comparison between RETE and our proposed algorithm with a rule (IF A = $\langle x\rangle$^ B = $\langle x\rangle\langle y\rangle$^ C = $\langle y\rangle$ THEN action) and the facts in Table 1. The original RETE tries to trigger the action with two 2-input nodes (Figure 6(a)) accompanying five comparison operations (One A node with three B nodes and one AB node with two C nodes) while RETE-ADH tries to trigger the action with simple referencing (Figure 6(b)) accompanying two operations (node A to B and node B to C). In this example, our proposed algorithm reduces the number of

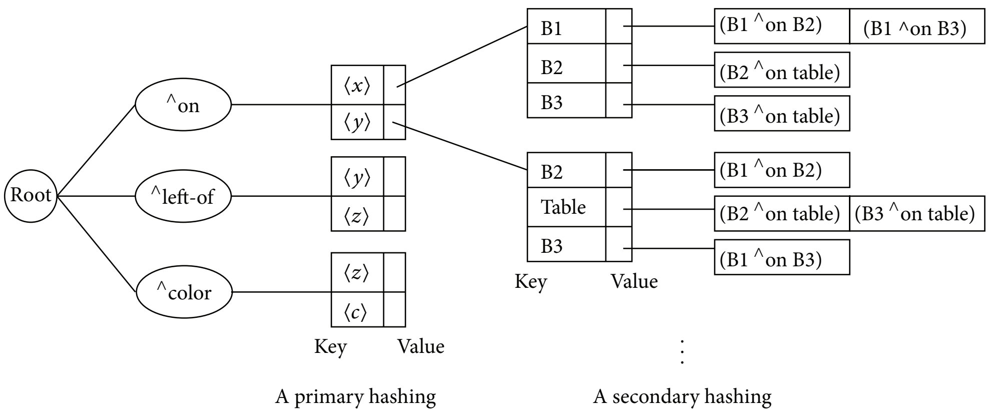


Figure 5: Alpha network of the RETE-ADH.

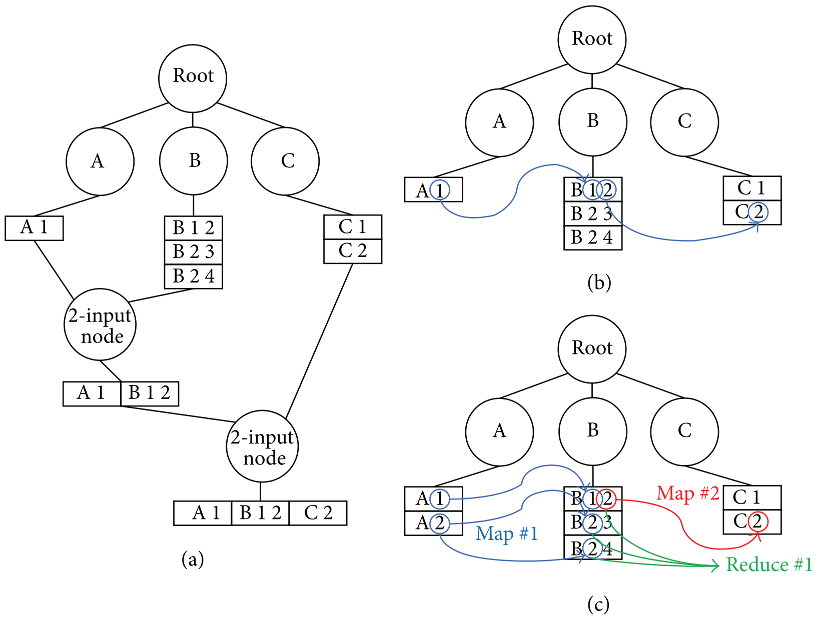


Figure 6: Comparison between RETE and RETE-ADH.

comparison operations by 60%, which roughly matches with the empirical results shown in Section 4.

Figure 6(c) shows another nice property of the proposed algorithm and its suitability for the parallelization. Assume that we use the same rule noted above. The tree in Figure 6(c) has one more node below *node* A compared to those in Figure 6(a) and Figure 6(b). Note that the *nodes* A, B, and C and each of the groups of nodes below them can be stored separately. For example, we can parallelize the algorithm by implementing the *map-reduce* computation. After receiving facts, we can execute a *map* phase in which the nodes that should be searched are selected. Then we have a blue group and a red group of *mappings*. After computations of these groups, we can execute a *reduce* phase, noted in green. Note that we can compute each of the grouped *mappings* concurrently. Although in the case of Figure 6(c), there is some overhead (relative to the sequential algorithm) because each of the computations in grouped *mappings* should be done thoroughly; we expect that for a large set of rules and facts, such a parallelized algorithm can easily outperform sequential ones. In addition, a parallel algorithm can delegate the effort required for the matching process to modern parallel processors such as CUDA GPUs, leaving the main CPU free to perform other tasks.

4. Test Bed: Virtual Simulator for a Smart Office Environment

In order to validate the proposed architecture and algorithm, we prepared a number of test cases targeting application in a smart office environment. Suppose that we are developing the smart office scenario depicted in Figure 7 with a virtual simulator. The imaginary smart office application automatically provides the most appropriate service for the employees by using the pattern matching algorithm. When an employee enters the office, the smart office application

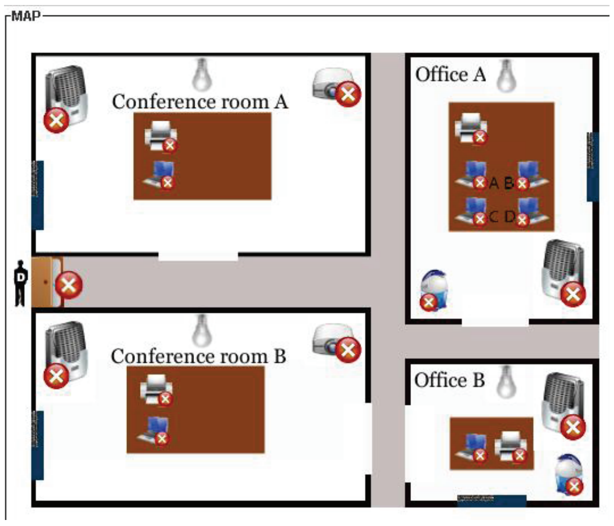


Figure 7: Composition of the smart office.

will automatically provide context-aware services such as turning on his/her computer, light, and printer. In addition, it maintains comfortable temperature, humidity, and illumination for the users by controlling air conditioners, humidifiers, and lamps, allowing employees to work in an optimal environment without bothering with environmental controls.

4.1. Overview. Let us assume that three employees enter office A and one employee enters office B. Before it can yield meaningful context information, the system will need to collect information as follows.

(i) Sensing: gathering context information from sensor devices.

(ii) Aggregating: observing, collecting, and composing context information from various context information processing units.

(iii) Inferring: interpreting context information to derive other types of context information, based on logic rules and the knowledge base, for instance.

(iv) Adopting: projecting context information of given situations.

One smart office scenario for a specific context adaptation case is demonstrated in Figures 8 and 9. After the user enters the smart office building, the user's device sends a request/subscription message to the composite context-aware system. Based on the device ID and user ID, the main system identifies the user and provides services accordingly.

Figure 8 shows a layout of the developed smart office application. Our smart office application mainly consists of three components: the MAP, DATA, and SYSTEM MESSAGE modules. An event controller is also developed, as shown in Figure 8. Each component is described as follows.

(i) MAP module: this module shows the office environment graphically. In this application, we assume that there are four places: conference room A, conference room B, office A, and office B. Also, this module depicts the state of each electronic device, door, and window, as well as each employee's current location.

(ii) DATA module: this module shows the office environment numerically; that is, through context information values. If a scenario occurs, this module shows the resulting changes in each context value.

(iii) SYSTEM MESSAGE module: this module shows the sequential control flow when a scenario operates. It chronologically shows the scenarios that have been occurring, the contexts that have changed, and the values of those contexts.

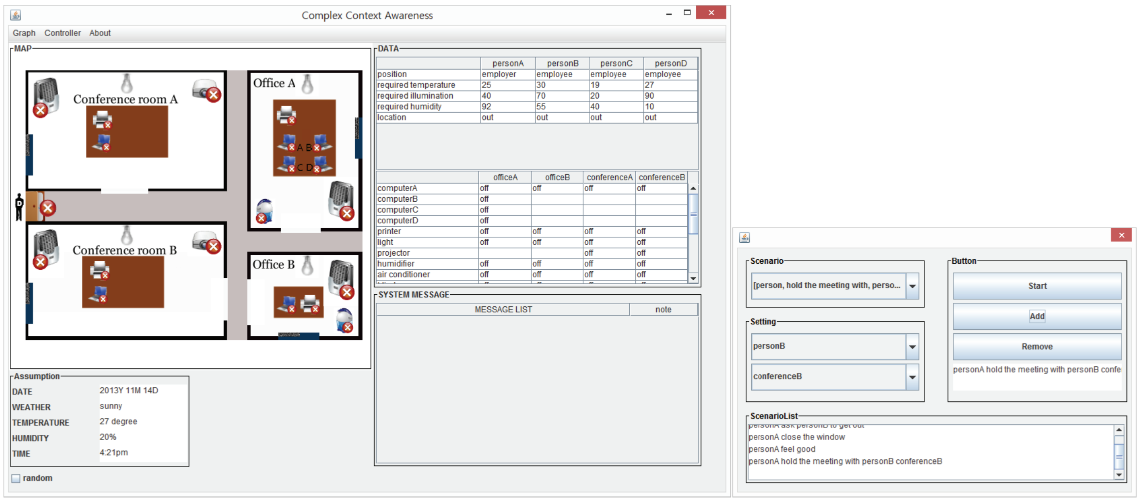
In this application, we can generate a virtual scenario by using the event controller. Table 2 shows the scenarios included in the smart office application. If we want to simulate any other scenarios in the smart office environment, new scenarios can be added using the event controller.

4.2. Structure. To validate the efficiency of the RETE-ADH algorithm in the composite context-aware system, we compared it with three other pattern matching algorithms in the virtual simulator. For each algorithm, the inference engine executed the algorithm and recorded the execution time for performance comparison. The *control package* integrated each algorithm. By using the *filemanage* function in the *control package*, each algorithm was identified and adopted. The related rules and facts were sent and received through this *filemanage function*. Each algorithm sent and received necessary parameters or call functions using the *AlphaNet.java*, *AlphaNode.java*, and *BetaNode.java* files.

Figure 10 shows a class diagram of RETE-ADH. When the RETE-ADH algorithm receives rules and facts, it differentiates the facts and records them to the alpha memory. After storing the facts, RETE-ADH composes the alpha network while considering their relationships. This process is performed by using the *NewRete*, *AlphaNet*, *AlphaSubNode*, and *AlphaNode* classes. The *AlphaSubNode* class is used only in the RETE-ADH algorithm, to support its secondary hashing. The *BetaNode* class is used for saving interim results. In the cases in which other algorithms are used, the basic structure and process is similar.

Figure 8 shows an empty smart office. In the *datamodule*, we can see the four employees' favorite temperatures, illumination levels, and humidity levels, as well as their respective locations. Note that, their locations are indicated as *Out* because none of them are present. Moreover, we can see that all devices are turned off. Now, we generate an event using the event controller: <*employee* A, B, C. *and* D *enter office* B,A, A, *and* A, *respectively*>.

In the virtual simulator, we follow ECA-DL; accordingly, this event can be expressed as person A goes to work at office B, person B goes to work at office A, person C goes to work at office A, and person D goes to work at office A. Figure 9 shows that all the employees have entered their offices. When the simulation begins with the start button on the event controller, the person icons move to offices



DATA

	personA	personB	personC	personD
position	employer	employee	employee	employee
required temperature	25	30	19	27
required illumination	40	70	20	90
required humidity	92	55	40	10
location	out	out	out	out

	officeA	officeB	conferenceA	conferenceB
computerA	off	off	off	off
computerB	off			
computerC	off			
computerD	off			
printer	off	off	off	off
light	off	off	off	off
projector			off	off
humidifier	off	off	off	off
air conditioner	off	off	off	off

SYSTEM MESSAGE

MESSAGE LIST	note

Assumption

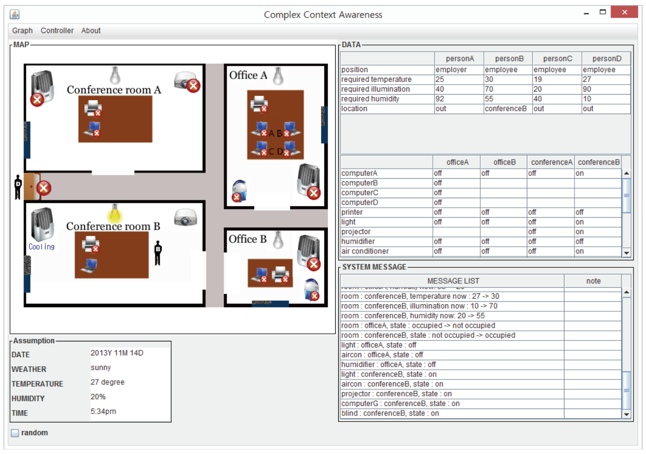
DATE	2013Y 11M 14D
WEATHER	sunny
TEMPERATURE	27 degree
HUMIDITY	20%
TIME	4:21pm

personA hold the meeting with personB confe

ScenarioList

personA ask personB to get out
personA close the window
personA feel good
personA hold the meeting with personB conferenceB

Figure 8: Initial status of the smart office.



DATA

	personA	personB	personC	personD
position	employer	employee	employee	employee
required temperature	25	30	19	27
required illumination	40	70	20	90
required humidity	92	55	40	10
location	out	conferenceB	out	out

	officeA	officeB	conferenceA	conferenceB
computerA	off	off	off	on
computerB	off			
computerC	off			
computerD	off			
printer	off	off	off	off
light	off	off	off	on
projector			off	on
humidifier	off	off	off	off
air conditioner	off	off	off	on

SYSTEM MESSAGE

MESSAGE LIST	note
room : officeA, humidity now: [illegible]	
room : conferenceB, temperature now : 27 -> 30	
room : conferenceB, illumination now : 10 -> 70	
room : conferenceB, humidity now: 20 -> 55	
room : officeA, state : occupied -> not occupied	
room : conferenceB, state : not occupied -> occupied	
light : officeA, state : off	
aircon : officeA, state : off	
humidifier : officeA, state : off	
light : conferenceB, state : on	
aircon : conferenceB, state : on	
projector : conferenceB, state : on	
computerG : conferenceB, state : on	
blind : conferenceB, state : on	

Assumption

DATE	2013Y 11M 14D
WEATHER	sunny
TEMPERATURE	27 degree
HUMIDITY	20%
TIME	5:34pm

Figure 9: Changed status of the smart office.

A and B. Furthermore, the devices are automatically set to each employee's favorites. In Figure 9, we can observe the changes in the states of each person and electronic devices through the DATA, SYSTEM MESSAGE and MAP modules. Three employees are located in office A and one employee is located in office B. Moreover, personal computers, lights, air conditioners, and humidifiers are turned on in both rooms A and B. In the DATA module, the employees' locations are set up, and other parameters are changed accordingly. However, the humidity, temperature, and illumination are not set to each employee's favorites. This is because the three persons in office A have different favorites.

4.3. Empirical Result. In this section, we compare the proposed scheme with the RETE, TREAT, and LEAPS algorithms in the context of smart office virtual simulator. In the smart office scenario, our smart office application tries to obtain context information by conducting pattern matching between a set of rules and fact information. Then the application picks appropriate services according to the context and provides them to the users.

For this simulation, we randomly generated various scenarios and measured the processing speed of each pattern matching algorithm, which indicates the average processing time to find a successful match.

Figure 11 shows the pattern matching performance for each algorithm. As shown in the figure, the LEAPS algorithm provided the best processing performance in finding a single rule. As noted previously, this is because the LEAPS algorithm does not try to find the full matched set, and fires at most one rule per matching cycle. Our proposed algorithm outperformed TREAT and RETE. As mentioned before, TREAT and RETE spend much time in constructing the network, which makes the corresponding systems access memory frequently whenever the fact information is updated. One interesting finding is that the RETE algorithm had better performance than the TREAT algorithm. In the smart office application, the number of rules that can be fired is relatively limited. It can be deduced that the RETE and RETE-ADH algorithms effectively utilize their beta memories in the given environment. Note that we cannot adopt the LEAPS algorithm for our application, which needs a full set of matched rules

Table 2: Scenarios included in the smart office application.

Scenario rule	Description
Person, go to work at, room	Commutes to the office
Person, go home, room	Get off work
Person, hold a meeting with, person, at room	Hold a meeting (two people)
Person, hold a meeting with, person, person, at room	Hold a meeting (three people)
Person, hold a meeting with person, person, person, at room	Hold a meeting (four people)
Person, adjourn the meeting with, person	Adjourn a meeting (two people)
Person, adjourn the meeting with, person, person	Adjourn a meeting (three people)
Person, adjourn the meeting with, person, person, person	Adjourn a meeting (four people)
Person, call, person	Call person
Person, call, person, person	Call two persons
Person, call, person, person, person	Call three persons
Person, ask, person, to leave	Ask person to leave
Person, ask, person, person, to leave	Ask two persons to leave
Person, ask, person, person, person, to leave	Ask three persons to leave
Person, open the window	Open the window
Person, close the window	Close the window
Person, open the blinds	Open the blinds
Person, close the blinds	Close the blinds
Person, use printer	Use a printer
Person, get tired	Get tired (abstract scenario)
Person, feel bad	Feel bad (abstract scenario)
Person, feel good	Feel good (abstract scenario)
Person, happy	Happy (abstract scenario)
Person, unhappy	Unhappy (abstract scenario)

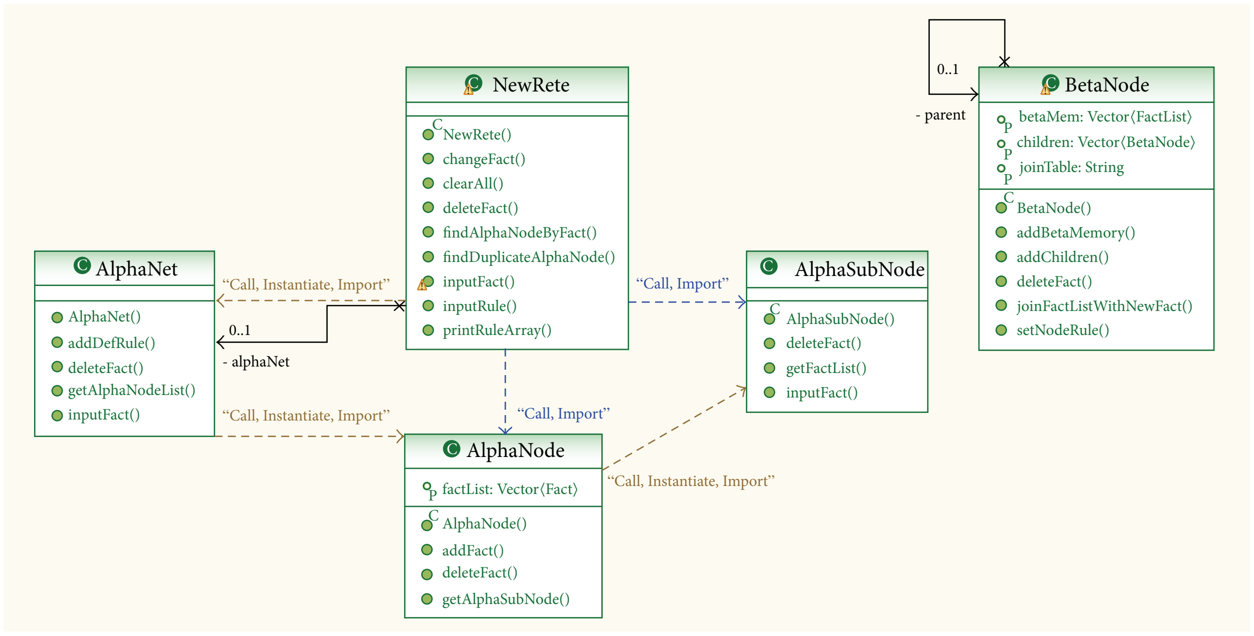


Figure 10: Class diagram of the RETE-ADH. (NewRete class represents RETE-ADH class).

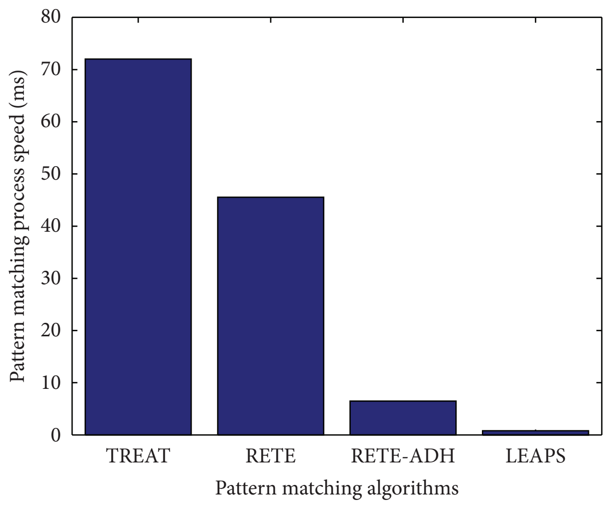


Figure 11: Experimental results.

to signify accurate context, and to enable the system to provide the most appropriate services. Rejecting the LEAPS algorithm leaves us with the options of TREAT, RETE, and our proposed algorithm. The experimental results suggest that our proposed algorithm is the best fit among these options for the targeted composite context-aware service.

5. Conclusions and Future Work

In this paper, we proposed a composite context-aware service architecture and a new pattern matching algorithm. In addition, we implemented a virtual simulator to validate our architecture and algorithm. The proposed algorithm provides enhanced matching performance by searching only a subset of the rules that can be matched. This improvement was made possible by the adoption of double hashing in the alpha network. We compared the proposed algorithm with the well-known pattern matching algorithms RETE, TREAT, and LEAPS by using our virtual simulator. The simulation results show that our proposed algorithm outperforms the TREAT and RETE algorithms. In addition, LEAPS was rejected due to its unique behavior of firing at most one rule per matching cycle, which is insufficient for context aware services. It was observed that the matching performance of the proposed algorithm was improved by 85% compared to that of RETE. We presented a practical scenario set in a smart office to show the applicability and validity of our composite context-aware service architecture.

In the future work, we will extend the proposed algorithm to exploit a parallel hardware architecture such as that of a CUDA GPU. In addition, we plan to carry out experiments using actual sensor nodes in various real-world scenarios.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was supported by the Chung-Ang University Excellent Student Scholarship, the National Research Foundation (NRF-2011-0013924) and the MSIP (Ministry of Science, ICT and Future Planning) under the ITRC (Information Technology Research Center) support Program (NIPA-2013-H0301-14-1023) supervised by the NIPA.

References

[1] L. M. Daniele, P. D. Costa, and L. F. Pires, "Towards a rule-based approach for context-aware applications," in *Proceedings of the 13th Open European Summer School and IFIP TC6.6 Workshop on Dependable and Adaptable Networks and Services*, Lecture Notes in Computer Science, Springer, Enschede, The Netherlands, 2007.

[2] A. Zaslavsky, "Mobile agents: can they assist with context awareness?" in *Proceedings of the IEEE International Conference on Mobile Data Management (MDM '04)*, pp. 304–305, January 2004.

[3] C. L. Forgy, "Rete: a fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, 1982.

[4] M. Matsushita, M. Umano, I. Hatono, and H. Tamura, "A fast pattern-matching algorithm using matching candidates for production systems," in *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence*, Lecture Notes in Computer Science, Springer, Cairns, Australia, 1996.

[5] D. P. Miranker, "TREAT: a better match algorithm for AI production systems," in *Proceedings of the 6th National Conference on Artificial Intelligence*, pp. 42–47, 1987.

[6] D. P. Miranker and B. J. Lofaso, "The organization and performance of a TREAT-based production system compiler," *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 1, pp. 3–10, 1991.

[7] D. Batory, "The LEAPS algorithms," Tech. Rep., University of Texas at Austin, Austin, Tex, USA, 1994.

[8] D. Liu, T. Gu, and J.-P. Xue, "Rule engine based on improvement rete algorithm," in *Proceedings of the International Conference on Apperceiving Computing and Intelligence Analysis (ICACIA '10)*, pp. 346–349, Chengdu, China, December 2010.

[9] D. Xiao and X. Zhong, "Improving rete algorithm to enhance performance of rule engine systems," in *Proceedings of the International Conference on Computer Design and Applications (ICCDA '10)*, pp. V3572–V3575, Qinhuangdao, China, June 2010.

[10] D. Zhou, Y. Fu, S. Zhong, and R. Zhao, "The Rete algorithm improvement and implementation," in *Proceedings of the International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII '08)*, pp. 426–429, December 2008.

[11] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, Lecture Notes in Computer Science, Springer, London, UK, 1999.

[12] W. Na, S. Cho, E. Kim, and Y. Choi, "Event detection in composite context aware-service," in *Proceedings of the 3rd International Conference on Ubiquitous and Future Networks (ICUFN '11)*, pp. 342–345, Dalian, China, June 2011.

[13] J. McDermott, A. Newell, and J. Moore, "The efficiency of certain production system implementations," *ACM SIGART Bulletin*, no. 63, pp. 38–38, 1977.

[14] K. Oflazer, "Highly parallel execution of production systems: a model, algorithms and architecture," *New Generation Computing*, vol. 10, no. 3, pp. 287–313, 1992.

[15] D. A. Brant, T. Grose, B. Lofaso, and D. P. Miranker, "Effects of database size on rule system performance: five case studies," in *Proceedings of the 17th International Conference on Very Large Data Bases*, pp. 287–296, San Francisco, Calif, USA, 1991.

[16] J. Yoon and K. Chung, "An efficient pattern matching alogorithm for AI production system," in *Proceedings of the Korean Institute of Information Scientists and Engineers (KIISE '94)*, 1994.