Composite Context-Awareness Service Architecture and Virtual Simulator with Efficient Pattern Matching Algorithm

Gunwoo Lee\textsuperscript{a}, Woongsoo Na\textsuperscript{a}, Changun Lee\textsuperscript{a}, Sungrae Cho\textsuperscript{a,*}, Eungha Kim\textsuperscript{b}, Youngil Choi\textsuperscript{b} and Sukyung Kim\textsuperscript{b}

\textsuperscript{a}School of Computer Science and Engineering, Chung-Ang University, 221 Heukseok, Seoul 156-756 South Korea
\textsuperscript{b}Cognitive Service Control Research Team, Service Platform Research Department, Electronics and Telecommunications Research Institute, 138 Gajeongno, Yuseong-gu, Daejeon, South Korea
E-mail: {gwlee,wsna}@uclab.re.kr, {cglee,srcho}@cau.ac.kr, {ehkim,yichoi,ksk0314}@etri.re.kr

Abstract. We propose an architecture for composite context-awareness service and a new pattern matching algorithm. The proposed architecture enables flexible integration of various context-awareness services. The new algorithm extends RETE to effectively support the systems based on our architecture. It provides reduced matching time by searching only subset of the rules that can be matched. In addition, a virtual simulator is implemented for validating our architecture and algorithm. Our experiment result shows that the proposed algorithm outperforms three well-known pattern matching algorithms, RETE, TREAT, and LEAPS. In the simulation, the matching speed and accuracy of the proposed algorithm was improved by about 85% and 8% respectively compared to RETE. A practical scenario of smart office is presented to show the applicability and validity of our approach.

Keywords: Composite context awareness, pattern matching algorithm, RETE, TREAT, LEAPS

1. Introduction

In the past few years, the cost of wireless personal devices such as smart phone has been dropping while the capabilities of these devices continue to increase. Interconnecting these computing devices provide intelligent services along with various sensing technologies regardless its size, from simple motion sensor to electronic tags or to video cameras [4]. Furthermore, advances in software technology, computing devices, and the increasing volume of digital knowledge offer the opportunity for more sophisticated and user-friendly digital services [15] such as ubiquitous computing. They can be envisioned as a form of data communication between entities that do not necessarily need human interaction. They are different from current communication models as they involve new or different market scenarios, low costs and effort, a potentially very large number of communicating terminals, and small and infrequent traffic transmission per terminal.

At the same time, a concept of “context-awareness” is suggested with ubiquitous computing. Context-awareness technique is used for services that provide proper and intelligent application to the user by accepting context information according to the user’s environment. Context can be considered as a set of information that includes user’s activity, location, personal preferences and current status while the most widely accepted formal definition has been provided by Dey and Abowd [1]: "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

*Corresponding author. E-mail: srcho@cau.ac.kr.
In the previous study [11], we classify contexts into unitary and composite contexts. We define the unitary context is a basic building block that is not further dividable. Also, as shown in Fig. 1, we define the composite context information as enhanced high level context abstraction by integration or composing unitary context information related with multi-entities to decide or supply the service of system [11]. In order to provide composite contextual service, multiple unitary contexts are combined as shown in Fig. 1. For example, suppose a tourist is planning to visit some attraction site near his home. Then, visiting attraction site near home would be considered as composite context. To construct the composite context, the following various unitary contexts will be related to the composite context: weather, allowed time, distance to the site, budget, accompanying people, transportation, etc. Future context-aware service is expected to form such composite context information. While providing a usable context to users is important, it is also essential to reserve such a composite context even if it is not available.

To support this composite context-awareness service, event-control-action (ECA) architecture can be used. The core algorithm which is used in the inference engine of the ECA is the most important part since it decides system performance. In this reason, we analyze existing algorithm of the inference engine such as RETE [5], TREAT [9], and LEAPS [2]. The RETE algorithm searches thoroughly all matched rules. In this reason, matching time can be considerably large while the accuracy performance is the best among the three. The TREAT algorithm provides comparable accuracy with reduced pattern matching time by removing beta memory. The LEAPS algorithm, on the other hand, computes at most one match per cycle. Therefore, it provides relatively smaller time to provide matched rule. However, it may provide wrong result.

In this paper, we propose a composite context-awareness service architecture that can detect user’s high level context abstraction and then provide a useful service to the user. The proposed composite context-awareness service architecture relies on pattern matching algorithm. To tackle the existing pattern matching algorithms, we propose a new pattern matching algorithm referred to as RETE Alpha-network dual hashing (RETE-ADH) in this paper. The proposed RETE-ADH algorithm searches only a part of all executable rules by reconstructing alpha network with hash table. In other words, RETE-ADH algorithm chooses facts which have high possibility of the matching. This can reduce pattern matching time significantly by making relationships between alpha and beta memory while provide comparable accuracy performance as RETE. Furthermore, the proposed composite context-awareness service architecture is demonstrated with a virtual simulator in a smart office.

The remaining of this paper is organized as follows. Composite context-awareness service architecture is explained in section 2. In section 3 and section 4, ECA architecture and existing pattern matching algorithms for the ECA are discussed, respectively. The proposed RETE-ADH algorithm is explained with examples in section 5. Section 6 introduces virtual simulator for smart office environment. Section 7 evaluates performance of the RETE-ADH algorithm with the existing pattern matching algorithms in smart office simulator. Finally, we draw conclusions and suggest future directions in section 8.

2. Composite Context-Awareness Service Architecture

Fig. 2 shows our proposed composite context-awareness service architecture. When user sends his/her composite context to the composite context-awareness
service, the service architecture requests/subscribes the composite context information (CCI) through service layer. The user requests the CCI to the context-awareness service, then the service responds to his/her request if the service finds the proper presence information immediately from the CCI interface control function. Otherwise, the user subscribes the CCI to the context-awareness service. Then, the composite context-awareness service processes the CCI by underlying CCI interface control function. Once the CCI interface control function finds a proper presence information of the context, the information is notified 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 to the later request/subscription.

In the CCI interpretation function, the request/subscribe command is differentiated by the 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 the rule pattern matching algorithm, the inference engine finds a proper presence information of the context. In the inference engine, we adopt our proposed pattern matching algorithm. By using CCI repository, proper matching algorithm can facilitate the matching speed.

Fig. 3 shows an example of the CCI that is processed in the proposed composite context-awareness service architecture. The context-awareness service should provide to the user with high capability smart devices in home, office, shopping mall, etc. In other words, the composite context-awareness applications that can monitor such regions and they can improve human’s lifestyle. Based on this, the CCI needs to be expanded with user’s current environment. For example, the CCI is expanded using device ID, user ID, service ID, and location ID. The device can have its unique parameters such as device name, type, provider, supported service, connected network status, its location, its owner, etc. In the similar fashion, the user ID has its unique parameters such as user name, device name, user’s location, user’s status, etc.
3. Rule-based System composing ECA

The composite context-awareness system is based on ECA pattern. The ECA pattern is composed of three modules such as event, control, and action based on the concept of condition and rule. The rule is expressed as IF <condition> THEN <action>. The <condition> part of the rule specifies the situation under which the actions are enabled, and it is composed by logical combination of events. Events model some occurrence of interest in application or environment. The <action> part of the rule is composed of one or more actions that are triggered whenever the <condition> part is satisfied [4].

In the rule-based system, the ECA performs a proper action when the context information is entered. In the rule-based system, fact and rule are core concept. Fact is an expression of a certain situation or environment in real world. Rule is a decided method how the system can behave in a certain situation.

Rule-based system is composed of working memory, knowledge base, inference engine, and action performer. The working memory is a space which saves fact. This fact is frequently updated and changed. The knowledge base is a space which saves rules. The rule tests the conditions under the criteria of the rules. The inference engine is a core part of the rule-based system. The inference engine checks the facts whether it satisfies rules or not based on substitution method at one cycle of the inference engine. This test is adopted all of the rules. Therefore, the number of calculations at one cycle of the inference engine is almost tens of thousands. This process of the calculation is called pattern matching. After deciding a rule at last, the <action> part is executed, and this process is called "fire." When rule is fired, the facts of the working memory can be updated. The pattern matching time consumes almost of whole processing time. In this reason, if the pattern matching time can be reduced, the performance of whole system can be improved. To improve the pattern matching process, several pattern matching algorithms including RETE [5], TREAT [9], and LEAPS [2] have been proposed. We will discuss about these algorithms in section 4. The inference engine which uses RETE or TREAT algorithm has a space called agenda. The agenda temporarily saves rules whose <condition> part matches the facts. These rules are called conflict set.

4. Pattern Matching Algorithm

In this section, we will give a brief overview of RETE, TREAT, and LEAPS, which are the most pop-
4.1. RETE

Most of the pattern matching algorithms save the partial matches produced during the previous inference cycle into their working memory. Thus, they can avoid re-evaluating the whole facts whenever the changes are made.

RETE algorithm [5] is one of such pattern matching algorithms and it maintains a network of partial matches for its run-time efficiency. This network is composed of the nodes containing 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/deleted to/from working memory.

There have been many previous efforts addressing the problems of the RETE [7,9,10]. Among them, we specifically consider the issue of beta memory explosion. The RETE network has 2-input nodes, which is referred to as beta nodes. They send their outputs to beta memory. Since the RETE retains partial matches for performance reason the 2-input nodes increases rapidly, which again consumes enormous beta memory.

4.2. TREAT

One of the main goals of TREAT algorithm is to overcome a major problem of the RETE, i.e., to reduce the overhead of network management. This algorithm is motivated from 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” [8]. TREAT does not use beta memory unlike RETE, thus it reduces the overhead of network management significantly. As a result, TREAT exhibits maximum 50% better performance than RETE [9].

In order to avoid using the beta memory, TREAT re-computes the matches repeatedly [12]. This means that the job of finding the corresponding alpha nodes for the input of beta nodes is done repeatedly. It may render TREAT inappropriate for bounded algorithms.

4.3. LEAPS

Unlike RETE and TREAT, which use eager evaluation techniques, LEAPS adopts lazy evaluation. Instead of generating all possible matches, LEAPS computes at most one match per cycle. It discards the conflict set and use a stack structure instead. Thus, it can contribute to reduce the computing time compared to the case of RETE and TREAT which need conflict resolution processes. Furthermore, LEAPS does not need beta memory. The stack management cost for LEAPS is known to be very low for most applications. LEAPS shows better performance than other algorithms especially when the conditions are complex [3].

In spite of the advantages of LEAPS, the property that LEAPS produces at most one match per cycle hinders the adoption of the algorithm for our application purpose. In our application, an appropriate ser-
vice needs to be invoked by the composite context-awareness, which needs full set of match instances. J. Yoon et al. notes that the LEAPS may not be appropriate for some applications [14].

5. Proposed Algorithm

As mentioned before, RETE may consume a lot of resources due to its heavy use of beta memory. TREAT requires less memory than RETE while it may undergo explosive amount of re-computations. LEAPS provides better time and storage-wise efficiency than the others, but it may not fit to some applications, e.g. our own scenario where at most one match output is not enough. For this reason, we propose a new pattern matching algorithm called RETE-ADH which extends RETE algorithm with hashing techniques.

One of the primary features is to adopt the double hashing in the alpha network, thus RETE-ADH algorithm improves the matching speed. The adoption of the double hashing also contributes to reduce the number of beta nodes, which, reduces the volume of the conflict set. Hence it improves the execution time of the algorithm.

Recently, there have been a few approaches using alpha network hashing [6,13]. In the alpha network, the alpha nodes are used to evaluate literal conditions of the facts. The fact data will propagate 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 object-type node to alpha node [6,13]. In their research, an alpha node is added to a type-node, and it adds the literal value as a key to the alpha node.

In our proposed algorithm, we use double hashing as follows: Each alpha node is hashed to variable nodes. Each variable node consists of a variable name and a secondary hash table. Each entry in the secondary hash table consists of a pair of a fact attribute and a list of the related facts.

Note that the previous approaches using the alpha network hashing [6,13] have to search all the facts in the alpha network to build beta network. In contrast, the proposed algorithm does not choose useless alpha node by using the secondary hashing table. For this reason, the RETE-ADH can reduce the volume of the beta network.

Assume that we have the rules as shown in Fig. 4 (a). In Fig. 4, the rule searches for 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, \(<x>\) indicates a variable \(x\) in the condition. Constants and identifiers are not enclosed by the brackets. In order to fire this rule we need facts satisfying the three conditions, which we will refer to each as \(c_1\), \(c_2\), and \(c_3\).

Assume that we have a fact \((b_1 \hat{=} \text{ on } b_2)\) which satisfies the condition \(c_1\). Then, the fact satisfying the condition \(c_2\) will be \((b_2 \hat{=} \text{ left-of } b_3)\) since \(b_2\) in the condition \(c_1\) and \(b_2\) of the condition \(c_2\) should be matched. The fact satisfying the condition \(c_3\) will be \((b_3 \hat{=} \text{ color red } <c>)\). The \(<c>\) of the condition specifies a certain color that the user wants. Similarly, we can list the matched conditions as shown in Table 1. The RETE pattern matching algorithm constructs alpha network with Table 1 as shown in Fig. 4 (b).

In our study, we choose to extend RETE among RETE, TREAT, and LEAPS because RETE fits to our application purpose the best. If we use TREAT for our application, the recursive matching calculation of the TREAT can be explosive since there are huge number of facts in the composite context environment. We determine that LEAPS does not fit either because it produces at most one match per cycle and we cannot choose the most appropriate service from it alone.

Fig. 5 shows alpha network of the RETE-ADH based on the rule example shown in Fig. 4 (a). The matching process of RETE-ADH is similar to that of RETE. The difference is how to construct the alpha network. The RETE chooses one node of the alpha network and it tries to match with the nodes of the beta network by using all facts. In contrast, RETE-ADH algorithm chooses facts in the alpha network which have high possibilities of the matching with beta nodes. Assume that we try to match \(c_1\) with \(c_2\). In this case, \(c_1\) has two variables \(<x>\) and \(<y>\), \(c_2\) has two variables \(<y>\) and \(<z>\). Because of the hashing table is composed with the variables, RETE-ADH tries searching with these variables. Since \(c_2\) has no \(<x>\), RETE-ADH searches \(c_2\) with \(<y>\) of the \(c_1\). The \(c_2\) has an identifier \(^{\hat{=}} \text{ left-of}\), and \(c_2\) has a hashing table which sets \(<y>\) and

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facts of the (w_n), where (n \in {1, 2, ..., 9})</td>
</tr>
<tr>
<td>(w_1): (B1 (^{\hat{=}}) on B2)</td>
</tr>
<tr>
<td>(w_2): (B1 (^{\hat{=}}) on B3)</td>
</tr>
<tr>
<td>(w_3): (B1 (^{\hat{=}}) color red)</td>
</tr>
<tr>
<td>(w_4): (B2 (^{\hat{=}}) on table)</td>
</tr>
<tr>
<td>(w_5): (B2 (^{\hat{=}}) left-of B3)</td>
</tr>
<tr>
<td>(w_6): (B2 (^{\hat{=}}) color blue)</td>
</tr>
<tr>
<td>(w_7): (B3 (^{\hat{=}}) left-of B4)</td>
</tr>
<tr>
<td>(w_8): (B3 (^{\hat{=}}) on table)</td>
</tr>
<tr>
<td>(w_9): (B3 (^{\hat{=}}) color red)</td>
</tr>
</tbody>
</table>
a primary hashing  a secondary hashing

Fig. 5. Alpha network of the RETE-ADH algorithm.

<z> as a key. Each entry in the table of \(c_2\) consists of a pair of a fact attribute and a list of the related facts. In Fig. 5, B2 and B3 are substituted to the <y> of the \(c_1\). Therefore, we can find a fact (B2 \(^\wedge\) color blue) by using the primary hashing table which sets <y> of \(c_2\) as a key and the secondary hashing table which has B2 as a key. In this manner, we can find (B3 \(^\wedge\) color red) with B3. Note that RETE-ADH searches facts using the double hashing table instead of searching all of facts of the alpha node as shown in the above.

Let us briefly compare the proposed algorithm with RETE. In the previous example, RETE has to try 24 combinations of the facts to find matched condition in Fig. 4(b). In contrast, the RETE-ADH tries 4 combinations of the facts. In this specific example, the number of beta nodes is the same for RETE and RETE-ADH. Note that in case there are many conditions and facts, there will be huge number of beta nodes generated in RETE. Hence, the matching execution time of RETE is much worse than that of RETE-ADH in general.

6. Virtual Simulator for Smart Office Environment

In order to validate the proposed architecture and algorithm we prepared a number of test cases targeting for the application of smart office environment.

Suppose we are developing the smart office scenario presented in Fig. 6 with the virtual simulator. The imaginary smart office application automatically provides the most appropriate service for employees by using the pattern matching algorithm. When an employee enters his office, he does not need to turn on his computer, light, etc. since the smart office application provides these services automatically. In addition, it provides appropriate temperature, humidity, and lightness for the users by controlling air conditioner, humidifier, and lamp. Therefore, employees can work in optimal environment without bothering with their environment.

Let us assume that three employees enter the office A and one employee enters office B. Then, the system
need to collect some information as shown below before yielding meaningful context information.

- **Sensing:** Gathering context information from sensor devices.
- **Aggregating:** Observing, collecting and composing context information from various context information processing unit.
- **Inferring:** Interpretation of context information in order to derive another type of context information. Interpretation may be performed based on, for instance, logic rules, and knowledge base.
- **Adapting:** The projection of context information of given situations.

A demonstration of the running application for the smart office scenario for a specific context adaptation case is depicted in Fig. 7 and Fig. 8. After the user enters the smart office building, the user’s device sends request/subscription message to the composite context-awareness system. By looking into the device ID and user ID the main system identifies user and it provides a proper service to the user.

Fig. 7 shows a layout of developed smart office application. Our smart office application mainly consists of three components, MAP, DATA, and SYSTEM MESSAGE module. Additionally, event controller is developed as shown in Fig. 7. The description for each component is shown in the following:

- **MAP module:** Map module visually shows office environment. In this application, we assume that there are four places (conference room A, conference room B, office A, office B). Also, this part visually shows each electronic device’s state, door’s state, and window state. Further, this part also shows current employee’s location.
- **DATA module:** Data module numerically shows office environment i.e., it shows context information value. If a scenario is occurred, this part shows that each context value is changed by the scenario.
- **SYSTEM MESSAGE module:** System message module shows that the sequence flow when a scenario operates. It chronologically shows the occurred scenarios, changed contexts, and the values of contexts.

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

```java
fileManager = new FileManager();
fileManager.loadFact(fileFact); fileManager.loadRule(fileRule); switch(algorithm) { case 0: ret = new Rete(); for(int i = 0; i < fileManager.getRuleList().size(); ++i) ret.insertRule(fileManager.getRuleList().elementAt(i)); break;
  case 1: leaves = new LeapsAlgorithm(); for(int i = 0; i < fileManager.getFactList().size(); ++i) leaves.insertFact(fileManager.getFactList().elementAt(i)); break;
  case 2: treat = new Treat(); for(int i = 0; i < fileManager.getFactList().size(); ++i) treat.insertFact(fileManager.getFactList().elementAt(i)); break;
  case 3: newRete = new NewRete(); for(int i = 0; i < fileManager.getRuleList().size(); ++i) newRete.insertRule(fileManager.getRuleList().elementAt(i));
                  for(int i = 0; i < fileManager.getFactList().size(); ++i) newRete.insertFact(fileManager.getFactList().elementAt(i));
newRete.printRuleArray(); break;
}
```

![Fig. 9. Algorithm integration parts of the control package.](image)

To validate the efficiency of the RETE-ADH algorithm in the composite context-awareness system, we compared it with other three pattern matching algorithms in the virtual simulator as shown in Fig. 9. For each algorithm, inference engine executes its operation, and the accuracy and the execution time of the algorithm are recorded in the system for performance comparison. The control package integrates each algorithm. By using `filemanage` function in the control package, each algorithm is identified and adopted. The related rules and facts are sent/received through this `filemanage` function. Each algorithm sends/receives necessary parameters or call functions using `AlphaNet.java`, `AlphaNode.java`, and `BetaNode.java` files. The package of the each algorithm is shown in Fig. 10.

Fig. 11 shows a class diagram of the RETE-ADH. When RETE-ADH receives rule and fact, RETE-ADH differentiates the fact and records the fact to the alpha memory. After storing the fact, RETE-ADH composes alpha network while considering their relationships. This process is performed by using `NewRete`, `AlphaNet`, `AlphaSubNode`, and `AlphaNode` class. The `AlphaSubNode` class is used only in the RETE-ADH algorithm since RETE-ADH use a secondary hashing. The `BetaNode` class is used for saving interim results. In other algorithm cases, this basic structure and process operate similarly.
Fig. 7 shows an empty smart office. In the data module, we can see that four employees’ favorite temperature, illumination, humidity, and their location respectively. Especially, the location indicates *Out* since there is no one. Moreover, we can see that entire devices are turned off. Now, we generate event using the event controller as *<employee A, B, C, and D enter the office B, A, A, and A>* respectively.

In the virtual simulator, we follow ECA-DL. Therefore, the event can be expressed as: *a person A go to work at office B, a person B go to work at office A, a person C go to work at office A, and a person D go to work at office A*. Fig. 8 shows that all of employees enter their offices. When simulation begins with start button on the event controller, the person icon moves to the office A and B respectively. Furthermore, en-
Fig. 8. Changed status of the smart office.
Table 2
Provided Scenario in Smart Office Application

<table>
<thead>
<tr>
<th>Scenario Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>person, go to work at, room</td>
<td>Commutes the office</td>
</tr>
<tr>
<td>person, go home, room</td>
<td>Gets off the work</td>
</tr>
<tr>
<td>person, hold the meeting with, person, at room</td>
<td>Hold the meeting (two people)</td>
</tr>
<tr>
<td>person, hold the meeting with, person, person, at room</td>
<td>Hold the meeting (three people)</td>
</tr>
<tr>
<td>person, hold the meeting with person, person, at room</td>
<td>Hold the meeting (four people)</td>
</tr>
<tr>
<td>person, adjourn the meeting with, person</td>
<td>Adjourn the meeting (two people)</td>
</tr>
<tr>
<td>person, adjourn the meeting with, person, person</td>
<td>Adjourn the meeting (three people)</td>
</tr>
<tr>
<td>person, adjourn the meeting with, person, person, person</td>
<td>Adjourn the meeting (four people)</td>
</tr>
<tr>
<td>person, call, person</td>
<td>Call person</td>
</tr>
<tr>
<td>person, call, person, person</td>
<td>Call two persons</td>
</tr>
<tr>
<td>person, call, person, person, person, person</td>
<td>Call three persons</td>
</tr>
<tr>
<td>person, ask, person, to get out</td>
<td>Ask person to get out</td>
</tr>
<tr>
<td>person, ask, person, person, to get out</td>
<td>Ask two persons to get out</td>
</tr>
<tr>
<td>person, ask, person, person, person, to get out</td>
<td>Ask three persons to get out</td>
</tr>
<tr>
<td>person, open the window</td>
<td>Open the window</td>
</tr>
<tr>
<td>person, open the window</td>
<td>Close the window</td>
</tr>
<tr>
<td>person, close the window</td>
<td>Close the window</td>
</tr>
<tr>
<td>person, close the blind</td>
<td>Close the blind</td>
</tr>
<tr>
<td>person, open the blind</td>
<td>Use print</td>
</tr>
<tr>
<td>person, use print</td>
<td>Get tired (Abstract Scenario)</td>
</tr>
<tr>
<td>person, get tired</td>
<td>Feel bad (Abstract Scenario)</td>
</tr>
<tr>
<td>person, feel good</td>
<td>Feel good (Abstract Scenario)</td>
</tr>
<tr>
<td>person, happy</td>
<td>Happy (Abstract Scenario)</td>
</tr>
<tr>
<td>person, unhappy</td>
<td>Unhappy (Abstract Scenario)</td>
</tr>
</tbody>
</table>

Fig. 11. Class diagram of the RETE-ADH (NewRete class represents RETE-ADH class.)

The tire devices are set depend on the employee’s favorite feature. In Fig. 8, we can observe that each person’s state and electronic devices’ state in the DATA module, SYSTEM MESSAGE module and MAP are changed. Three employees are located in the office A and one employee is located in the office B. Moreover, three
personal computer, light, air conditioner, and humidifier are turned on at both room A and B. In the DATA module, the employees’ location is set up depending on their locations and other parameters are changed accordingly. However, the humidity, temperature, and illumination are not set to each employee Favorites. This is because three persons in the office A have different favorites.

7. Performance Evaluation

In this section, we compare the proposed scheme with the RETE, TREAT, and LEAPS by using smart office virtual simulator. Table 3 shows a specification of the simulation environment. When a smart office scenario begins, the developed smart office application conducts pattern matching between set of rule and fact information, and provides appropriate service.

For simulation, we randomly generate scenarios and measure the pattern matching accuracy and process speed defined by

- **Accuracy**: the ratio of correct results to total execution.
- **Process Speed**: the average of process time for finding successful matching.

We measure the process speed by varying the pattern matching algorithm. Fig. 12 shows the process speed versus pattern matching algorithm. As shown in the figure, the LEAPS algorithm has the fastest process speed to find an appropriate rule. This is because LEAPS algorithm can fire a rule whenever it finds executable rule. However, TREAT and RETE algorithm have lower speed than other algorithms since they spend a lot of time for constructing a memory network. It means that the system frequently accesses the memory for inserting or removing whenever fact information is updated. One interesting finding is that the RETE algorithm has lower speed than TREAT algorithm. In smart office application, the number of rules which can be fired is relatively small and limited. Therefore, RETE and RETE-ADH algorithm can efficiently utilize beta memory.

Fig. 13 shows the accuracy of the pattern matching algorithm in the smart office application. As shown in the figure, TREAT, RETE, and RETE-ADH algorithm shows similar accuracy for matching, and the accuracy of the LEAPS algorithm shows the lowest accuracy. This is because the LEAPS algorithm finds executable rules based on time stamps. Moreover, in the randomly chosen scenario, the frequency of the fired rule is also randomly distributed. It means that the LEAPS finds the correct rule rarely. However, other three algorithms construct memory network and they compare rules for finding appropriate rule. Especially, RETE-ADH al-

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>i7 sandy bridge</td>
</tr>
<tr>
<td>RAM</td>
<td>4GB</td>
</tr>
<tr>
<td>OS</td>
<td>Windows 7</td>
</tr>
<tr>
<td>JAVA Version</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Table 3

Specification of the simulation environment
The proposed algorithm provides reduced matching time by searching only subset of the rules that can be matched. This improvement was made possible by the adoption of double hashing in alpha network. We compared the proposed algorithm with well-known pattern matching algorithms, RETE, TREAT, and LEAPS by using our virtual simulator. The simulation result shows that our proposed algorithm outperforms other three schemes. It was observed that the matching speed and accuracy of the proposed algorithm was improved by about 85% and 8% respectively compared to RETE.

A practical scenario of smart office was presented to show the applicability and validity of our composite context-awareness service architecture.

Acknowledgements

This work was supported by the R&D program of KCC/KCA. [10913-05005, Research and development of composite context based adaptive service path configuration technology]

References


