Customizable Context Detection for ECA rule-based Context-aware Applications

T. Nakagawa, C. Doi, K. Ohta, and H. Inamura

Research Laboratories, NTT DOCOMO, Japan
3-6, Hikari-no-oka, Yokosuka, Kanagawa, 239-8536 Japan, nakagawato@nttdocomo.co.jp

ABSTRACT

Since user context detection plays a key role in realizing context-aware applications, developers must be able to create service-specific context detection algorithms easily. To this end, we propose an Event-Condition-Action (ECA) rule specification and rule engine specialized for customizable user context detection. The application developer of a context-aware application can utilize our ECA rule to define an algorithm to process terminal log data and output service-specific user context.

The proposed ECA rule specification satisfies the functional requirements in that it covers *W4H*, which are the five semantic dimensions for context description, and that it provides methods to handle various data types including sensor data and terminal logs of smartphone middleware. In addition, composite contexts, which are derived from several context sources, are also supported. An evaluation of a prototype implemented on an Android-based smartphone yielded a maximum response time of 300 ms, and CPU load of less than 5 %.

Keywords: ECA rule, Context-aware application, Logging function, User context detection, Customization

1 INTRODUCTION

Context-aware services are being researched to realize a convenient and comfortable life by providing appropriate services or applications suitable to our ever-changing situation [1]. A wide range of context-aware applications have been proposed so far, including context-aware tour guides, smart rooms, museum applications, mobile-learning, healthcare, and assisted living applications [2][3][4].

Recent smartphones are paving the way to these context-aware applications by providing rich information streams gathered from various sensors and open middleware. For example, research is underway on tackling the problem of capturing user context from sensor-enabled smartphones [5][6][7]. In addition, user context can also be acquired from operation logs of system events issued by the open middleware of smartphones [8]. Thus, smartphones provide the ideal platform for collecting abundant information from which user context can be acquired.

We adopt an ECA rule-based approach to realize context-aware applications, because previous research has revealed that it is an effective way of developing context-aware applications [1]. ECA rules are composed of event, condition, and action. When an ECA rule is executed in an ECA rule engine, the pre-defined action is automatically performed in response to events if the stated conditions hold [3][9][10]. The key benefit of the ECA rule approach is that it makes it easy to describe human problem-solving knowledge as procedural tasks [1]

However, existing context-aware systems utilize only gross user context such as time and location, or domain-specific user context generated by dedicated systems. In other words, the existing approach fails to support the application developer in creating original context detection algorithms.

We consider fine-grained user context defined on a per-application basis can lead to value-added context-aware applications with more user satisfaction. Let's take the context-aware tour guide, for instance. In addition to typical user context of location data, developers want to utilize additional terminal data to acquire more detailed user context. One developer would prefer to provide a function to distribute tour information based on the user's plan for the day, which is registered in the scheduler application on the smartphone. In this case, user context is acquired as a combination of the current location and the user's probable location in the future. Another developer might utilize pedometer data to decide the best opportunity for presenting recommended resting spots. In this case, user context is acquired by combining current location and user's physical status such as "being tired after long walk". In this way, the user context that the developer wants to utilize depends on the requirement of the application.

The goal of our research is to realize a platform on which developers can easily create original context detection algorithms to realize advance context-aware applications. Though various data is available for detecting user context on state-of-the-art smartphones, writing the source code in C or Java for collecting system status or sensory data is a cumbersome process. By realizing a platform that eliminates the need for this, application developers can focus on the main task of defining the appropriate context required for the application.

The primary contribution of this paper is a powerful ECA rule specification that allows developers to define user context detection methods on a per-application basis. The proposed ECA rule specification satisfies three functional requirements. First, it covers *W4H*, which are the five semantic dimensions for context description. Second, it provides methods to handle various data types gathered for a certain time span. Third, it allows composite contexts, which are derived from several context sources.

An evaluation of a prototype implemented on an Android 2.3 smartphone shows that the platform satisfies our performance requirements. The resource consumption and response time for context detection are shown to be acceptable for the resource-limited mobile environment.

2 RELATED WORK

We adopt the ECA rule-based approach because it is generally suitable for describing context-aware applications. To check if the rule-based approach is a suitable option, a list of questions is provided in [1]. For example, ECA rules are suitable for replicating human problem-solving knowledge because they provide a framework for representing procedural knowledge. Another benefit of ECA rules is that they make it easy to accommodate incremental changes in knowledge, because they can be easily customized by changing the parameters, attributes, and elements.

However, existing ECA rule studies on context-aware applications have not focused on creating original context detection algorithms, because they assumed that the events are given. For example, events such as *UserArrivalEvent* or *RoomStatusChangeEvent* are provided from outside of the rule engine in [2]. In [11], RDF is utilized to describe rules to prioritize and classify environmental services that are suitable for user's context. This approach can't be applied to describe context detection by log analysis.

Other intriguing studies address the challenge of acquiring user context from raw sensor data [5][6][7]. In LifeMap [5], smartphone-embedded sensors are utilized to extract user context, both indoors and outdoors. In [6], smartphones with built-in accelerometers are utilized to recognize user's activity such as walking, running, and walking upstairs and downstairs. In [7], road crossing is recognized via the smartphone's accelerometer. However, it remains true that the existing ECA rule-based approach lacks flexibility to describe user context on a per-application basis.

The contribution of this paper is to allow a developer to define an algorithm to generate primary context as desired. This is realized by providing event and condition tags with functions to analyze log data, and also providing action tags with functions to issue primary context. In [12], the complex process of collecting and analyzing sensor data is hidden from the developer so that context-aware applications can be easily created. In addition, high-level context can be handled by combining different widgets to make composite widgets. Though our approach offers the same features, it provides more powerful customization to describe original context detection algorithms. In [13], the preprocessing phase builds measurement data arrays that contain a certain number of samples and calculates generic features for each time interval. The proposed method allows such preprocessing to be customized by ECA rules.

We realize an ECA rule specification for service-specific user context detection; it allows the use of various data sets. Thus, the proposed system allows developers to utilize sensor data and system events from smartphone middleware to define original user context. As indicated in MobileSens [8], user context can be captured more precisely by taking advantage of the diverse supplementary data available from smartphone middleware. Studies such as MyExperience [14] and LiveLab [15] tackle the challenge by collecting various terminal logs to acquire user context. Those studies support our idea that abundant data from sensors and open middleware are effective for user context detection.

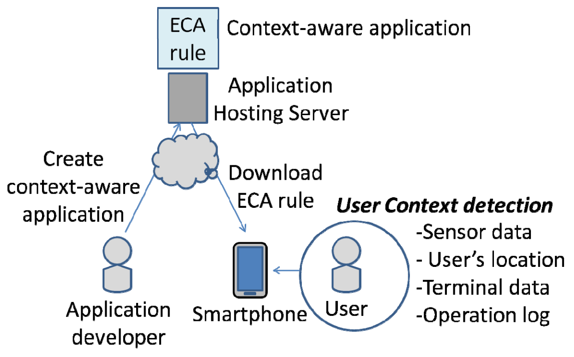


Figure 1: User context detection by ECA rule

The proposed ECA rule specification must provide flexibility in describing arbitrary user contexts regardless of the domain. From this view point, it is beneficial to refer to previous research on the semantics of context [16], and on composite context [17][18][19]. *W4H* [16] represents five semantic dimensions of identity (*who*), location (*where*), time (*when*), activity (*what*) and device profiles (*how*). Several studies have delved into the composite context, which is derived from several context sources [17][18][19]. In [17], complicated context is formed from unitary contexts such as permissible time periods, distance to the site, budget, etc. In [19], it is shown that the process-based approach is effective for creating composite context in a flowchart-like manner. We consider that both context semantics and composite context are essential elements in realizing domain-independent user context detection.

3 USER CONTEXT DETECTION

In this paper, user context detection means the process that user's situation is judged from various information such as sensor data, user's location, terminal data, and operation log (Figure 1). We introduce a mechanism that user context detection is conducted by the ECA rule, which is executed on an ECA rule engine on a smartphone.

Once a developer creates a context-aware application, the application is described using ECA rules. The event and condition part is the mechanism that captures user context by leveraging various terminal logs on the smartphone.

At first, an ECA rule engine is downloaded before utilizing context-aware applications, and executed as a resident program. Context-aware applications are downloaded on demand and executed on the ECA rule engine.

4 PROPOSED METHOD

4.1 Requirements

The following functional requirements must be satisfied to provide flexibility in defining original context detection. From the top-down viewpoint, the ECA rule specification should cover all conceivable user contexts. On the other hand, from the bottom-up viewpoint, various kinds of data gathered on the smartphone should be fully leveraged by ECA rule-based applications.

・**Functional Requirements**
1. Support for developing a context detection algorithm that covers the five semantic dimensions of *W4H*
2. Provision of a versatile method to process diverse types of log data gathered for different time spans
3. Support for processing composite contexts

Requirement 1 means that the ECA rule tags need to cover *W4H*, which are the five semantic dimensions of identity (*who*), location (*where*), time (*when*), activity (*what*) and devices profiles (*how*) [16]. Requirement 2 means that a wide range of log data must be processed properly including sensor data, communication logs, location data, and system logs. In addition, logs generated either continuously or intermittently, must be utilized for deep observation of user context. Requirement 3 is derived from previous research [17][18][19] which shows that the user context in the real world is often composed of several sub dimensions.

・**Performance Requirements**
4. CPU loads under 5% for rule engine execution
5. RAM consumption under 5MB
6. Response time of 300 ms for judging user context

Requirements 4 and 5 mean that even if various terminal logs are utilized by the application, the data gathering process should neither hinder other applications nor incur extreme resource consumption. Requirement 6 means that moderate response time must be assured for context-aware applications. The performance target of 300 ms is rather short compared to the evaluation results of previous studies on context-aware applications [9].

4.2 System Architecture

The proposed system is composed of a logging function, a log database, and an ECA rule engine (Figure 2). The logging function collects the terminal log data required for context detection and records the data to the log database. The rule engine executes the context-aware application using the log data read from the log DB.

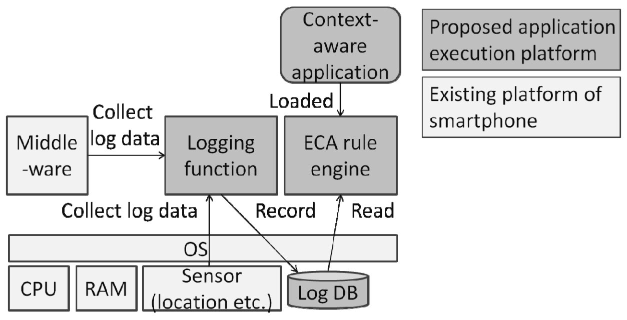


Figure 2: System configuration of the platform

The logging function collects log data from the smartphone's sensors and its middleware. The sensors provide user location, accelerometer data, and light level data, etc. The smartphone middleware provides terminal setting, terminal status and user operation. For instance, vibration mode setting, remaining battery level, backlight status, and applications utilized can be acquired from the middleware.

In order to protect the user's privacy, and also to assure rapid application response, the ECA rule engine is located on the smartphone. Because terminal logs are full of private data such as user location and operation logs, the data is processed only in the smartphone.

Table 1: Event and condition tags for context detection

Category	Tag name	Description
Who	OCCUR	Evaluate log generation concerning specific person by designating the target person
When	TIME	Evaluates time
Where	CENTER	Evaluates user location
What, How	OCCUR	Evaluates log generation
	RANGE	Evaluates log value (numerical comparison)
	MATCH	Evaluates log value (string comparison)
	SUM	Simple calculation on log generation count
	SUB	Simple calculation on log generation duration

4.3 Support for *W4H* in Context Detection

To satisfy requirement 1, each aspect of *W4H* is covered by the proposed ECA rule specification (Table 1). The context of *who* permits communication status analysis. For example, a context-aware application can grasp the situation that a phone call from a person has not been returned. The context-aware application can remind the user to call back and assist smooth communication.

As the ECA rule tag for describing *who*, OCCUR tag is utilized to detect log generation concerning specific people. For example, the caller in the incoming call log can be detected by the following description, where *log_ID_for* *incoming_call* means the log kind ID number that indicates an incoming phone call. Parameter “xxx-xxxx-xxxx” means the target phone number, which corresponds to *who*, in this log kind. As the terminal log is accompanied by several parameters, the parameter to be evaluated is specified by the POSITION attribute in the VALUE element. In this example, the phone number is the first parameter of the incoming phone call log.

```
<occur kind=”log_ID_for_incoming_call”>
  <value position=”1”>xxx-xxxx-xxxx</value>
</occur>
```

The dimensions of *when* and *where* are covered by TIME tag and CENTER tag, respectively. TIME tag supports repetitive events such as hourly or daily basis, in addition to the absolute designation of time. In the following example, the first line shows the absolute designation of midnight on 23rd May 2012, and the second line shows the repetitive designation of every hour. CENTER tag provides a function to detect user location, by specifying the center of the location represented by latitude and longitude, the positioning technology, and the radius of the target area. In the following example, the action is triggered when the user is within 1000 m of Naha city center.

```
<time><eq type=”datetime”>2012-05-23T00:00:00</eq>
<time><eq intervalType=”hour”>00:00</eq></time>

<center lat=”26.21662” lon=”127.688069” type=”gps”>
  <le type=”numerical”>1000</le>
</center>
```

The dimensions of *what* and *how* are covered by OCCUR tag and the remaining tags of RANGE, MATCH, SUM, and SUB, which also provide functions to satisfy requirement 2, see details in the next section.

4.4 Handling Diversity and Time Frame

To satisfy requirement 2, we assigned RANGE tag and MATCH tag to handle the different types of log data uniformly. RANGE tag is utilized to evaluate if the numerical data, such as sensor data or date data, enters into a specified range. MATCH tag is utilized to detect if specified string is included in the log. In the following examples, RANGE tag works to detect if a plan is scheduled in the specified period, while MATCH tag works to check if a plan of the specified title of “dinner” is scheduled. As MATCH tag supports string matching by the regular expressions of Perl, it provides powerful analysis of system logs and application logs.

```
 <range kind=”scheduler” position=”2”>
  <ge type=”datetime”>2012-5-23T18:00:00</ge>
  <le type=”datetime”>2012-5-23T19:00:00</le>
 </range>

 <match kind=”scheduler” position=”1”>dinner
 </match>
```

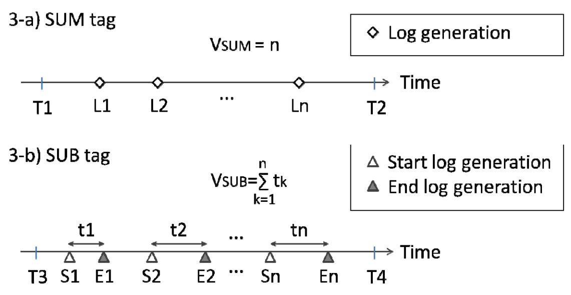


Figure 3: Evaluation of SUM tag and SUB tag

In addition, SUM tag and SUB tag are introduced to detect how often a certain behavior is performed by the user, or how long the user is dedicated to a certain activity, which is also claimed by requirement 2 (Figure 3).

SUM tag is evaluated by counting the total instances of terminal log generation in specified time span. In Figure 3-a, log generation instances from T1 to T2 are shown by symbols L1 to Ln. The evaluation result of SUM tag is provided by comparing the instance total, which is *n*, with the specified threshold. In the following example, the result is true if the number of steps from the pedometer data in the latest one hour is greater than or equal to 3000 steps.

```
 <sum kind=”pedometer” ge=”3000”>
  <trackback type=”hour”>1</trackback>
 </sum>
```

SUB tag is utilized to measure the total of each log generation interval specified by two kinds of different terminal logs that are generated alternately. In Figure 3-b, the start log and end log generation is shown by Sk and Ek. The evaluation result of SUB tag is provided by comparing the total of each duration tk, which is headed by Sk and terminated by Ek. In the following example, the result is true if browser usage of the day exceeds 60 minutes.

```
<sub from=“app_foreground” to=“app_background”
 ge=”60” type=”minute”>
  <eq type=”date”>today</eq>
  <gt type=”time”>00:00:00</gt>
  <value position=”2”>browser</value>
</sub>
```

4.5 Support for Composite Contexts

To satisfy requirement 3, we intoduce user-defined events as a kind of terminal log that can be freely defined by the developer.

Table 2: Action tags

Tag name	Description
LOGWRITE	Issues specified user-defined event as a terminal log.
DIALOG	Notify the user by showing message. Can hold LOGWRITE as child element, which is fired when a button is pressed.
COORDINATE	Issues an Android Intent to coordinate with the specified application.

Actions conducted after user context detection are defined by using action tags such as LOGWRITE, DIALOG, and COORDINATE (Table 2).

LOGWRITE tag is utilized to issue user-defined events recorded in the log DB for future use by event and condition tags. When LOGWRITE is executed on the rule engine, a terminal log called user defined event is recorded with the parameters defined in the EVENTVALUE tag. As the name "user-defined" indicates, the developer of the application can freely define multiple parameters.

```
<logwrite>
  <userevent>
    <eventvalue position=”1”>param1</eventvalue>
    <eventvalue position=”2”>param2</eventvalue>
  </userevent>
</logwrite>
```

The user-defined event can be handled by the event and condition tags in Table 1, as just another kind of terminal log. By combining event and condition tags and LOGWRITE tag, the developer can define an original method to detect user context from the terminal log, and record the detected user context in the log DB.

By using user-defined events, it is possible to detect composite contexts and thus create context-aware applications handling rich user context. To better understand how composite context can be detected, Figure 4-a shows the topology of the ECA rules. As the action can be executed after detection of several user contexts, ECA rules can be linked in a JOIN-type topology.

If several actions are executed after detection of a single context, the context detection and subsequent actions form a FORK-type topology (Figure 4-b). Both of these topologies are utilized in the evaluation of response time in Section 5.3.

DIALOG tag provides a simple function to show a dialog with predefined message and buttons.

Finally, COORDINATE tag activates other applications by the Intent mechanism of Android, which allows late runtime binding between different applications[1]. This function allows a context-aware application to trigger any other application to be executed.

[1] Intent | Android Developers
http://developer.android.com/reference/android/content/Intent.

4.6 Application and Customization Example

As a simple example, the XML source code of a context-aware tour guide is shown in Figure 5.

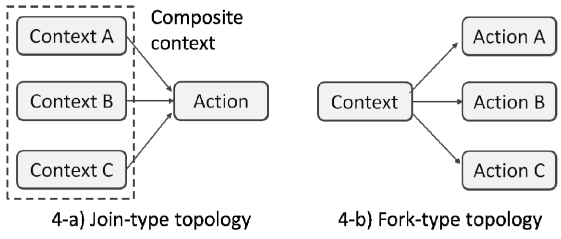
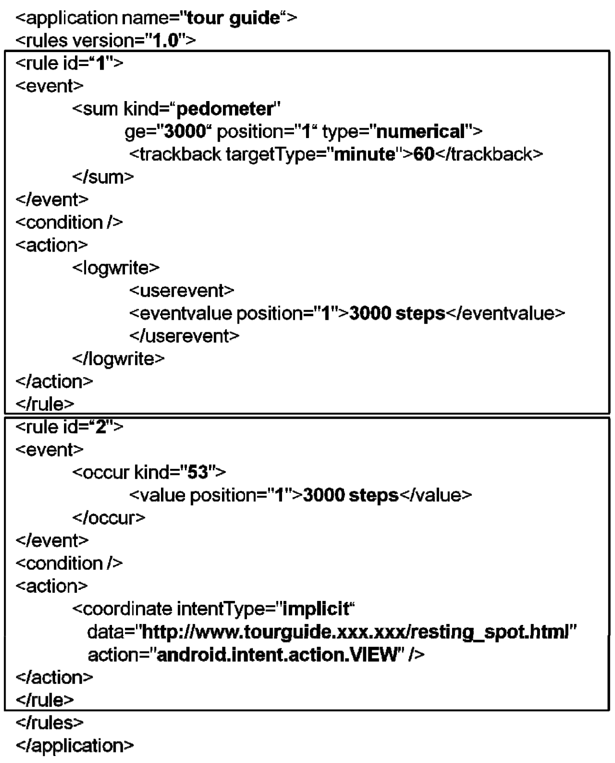


Figure 4: Evaluated topology of the ECA rules



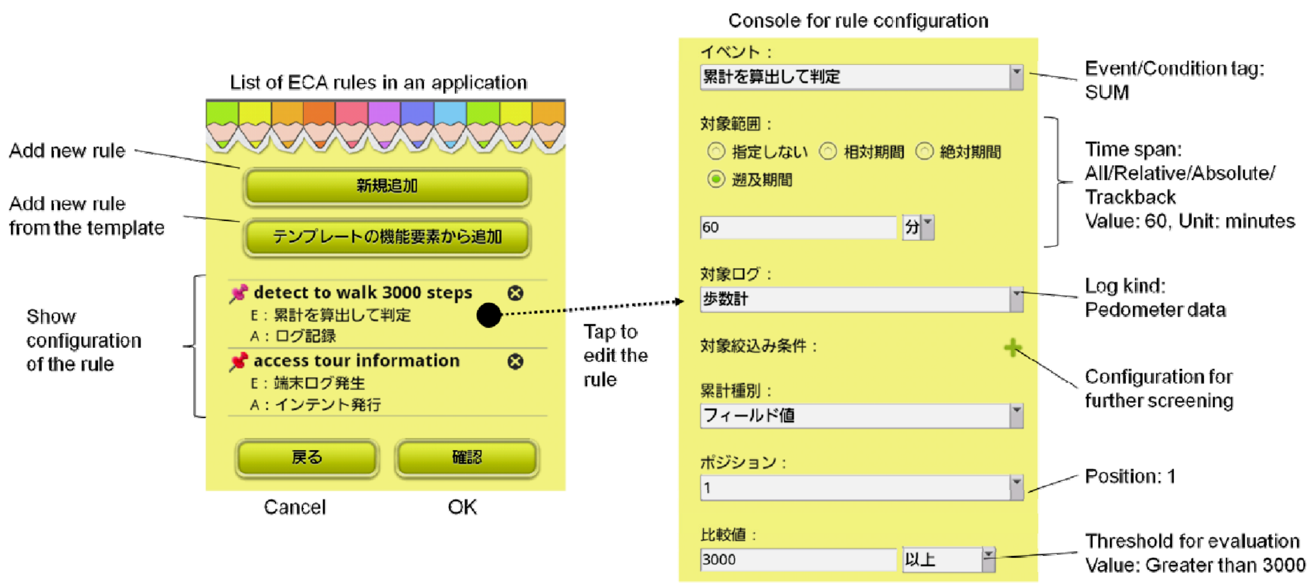
```
<application name="tour guide">
<rules version="1.0">
<rule id="1">
<event>
        <sum kind="pedometer"
                ge="3000" position="1" type="numerical">
                <trackback targetType="minute">60</trackback>
        </sum>
</event>
<condition />
<action>
        <logwrite>
                <userevent>
                <eventvalue position="1">3000 steps</eventvalue>
                </userevent>
        </logwrite>
</action>
</rule>
<rule id="2">
<event>
        <occur kind="53">
                <value position="1">3000 steps</value>
        </occur>
</event>
<condition />
<action>
        <coordinate intentType="implicit"
          data="http://www.tourguide.xxx.xxx/resting_spot.html"
          action="android.intent.action.VIEW" />
</action>
</rule>
</rules>
</application>
```

Figure 5: An example of context-aware tour guide

In this application, the user context of "long walk" is detected from pedometer data, and resting spots are shown by the web site automatically.

The ECA rule is composed of two rules, which define the detection of "long walk" and notification of the context. The first rule is fired when the total number of steps per hour exceeds 3000 steps. As a result, a user-defined event "***3000 steps***" is issued to record that context. The second rule is fired in response to the generation of the user-defined event of ***"3000 steps"***, and implicit Intent is issued to show the specific web site defined in the action tag.

The key feature of the proposed system is that the application is easy to develop by changing several configuration


List of ECA rules in an application
Add new rule
Add new rule from the template
Show configuration of the rule
新規追加
テンプレートの機能要素から追加
戻る
確認
Cancel
OK
Tap to edit the rule
Console for rule configuration
Event/Condition tag: SUM
Time span: All/Relative/Absolute/ Trackback Value: 60, Unit: minutes
Log kind: Pedometer data
Configuration for further screening
Position: 1
Threshold for evaluation Value: Greater than 3000

detect to walk 3000 steps
E：累計を算出して判定
A：ログ記録

access tour information
E：端末ログ発生
A：インテント発行

イベント：累計を算出して判定

対象範囲：指定しない 相対期間 絶対期間 [✓ 遡及期間]

60 分

対象ログ：歩数計

対象絞込み条件：

累計種別：フィールド値

ポジション：1

比較値：3000 以上

Figure 6: Screenshot of application development console

factors as provided by ECA rules. Below is a list of customization items for this scenario.

- **Parameters**

The goal of the number of steps may well vary. A developer might change the first parameter of pedometer log data from 3000 steps to 1500 steps.

- **Time span**

Instead of past 60 minutes, you can set longer or shorter time spans in the TRACKBACK element.

- **Additional conditions**

You can add other kinds of terminal log such as scheduler data to detect user context. For example, resting spots are shown only when the user doesn't plan to have dinner soon.

Also, user location can be considered to refine the tour information, by adding CENTER tag in the condition.

In the proposed method, a developer can define original context information necessary for context-aware applications with the minimum of customization via web browser console on a smartphone's or PC's browser (Figure 6). Hence, user context can be subdivided on the basis of the developer's original viewpoint. In other words, the developer can create context-aware applications more flexibly than previous development environments.

The configuration changes described above can be implemented by just web browser selections, and changing the ECA tags and parameters as desired. The customization can be completed even without writing an XML file of the ECA rule, because the ECA rule is automatically created according to the configuration input from the web console.

For example, if location context such as home or office is required by a context-aware application, the developer utilizes the web-based interface to input necessary data. The center and radius of the target are designated on the map, to specify the target's location and size. When the user of the application enters into or leaves from the target location, original context information such as HOME-IN or HOME-OUT is issued by executing an action defined by the developer. The developer is released from cumbersome procedures such as checking latitude and longitude data, and an ECA rule is automatically generated.

As this development console doesn't require programming skill, we assume it can be utilized by not only professional developers but also by end users.

5 EVALUATION

In this section, we evaluate the proposed system and confirm if the proposed ECA rule engine described in Section 4 satisfies performance requirements 4 through 6. The proposed system was implemented on an Android-based smartphone, whose specification is shown in Table 2.

The evaluation is conducted from the viewpoints of resource consumption, response time for single rule execution,

Table 2: Specification of the target smartphone

CPU clock	RAM	Flash ROM	Android version
1.0GHz	512MB	1024MB	2.3.4

and response time for FORK and JOIN type topologies. The resource consumption is measured in Section 5.1 to confirm if requirements 4 and 5 are satisfied when the ECA rule is processed in the rule engine. The response time is measured in Section 5.2 to confirm if requirement 6 is satisfied when time consuming tags are evaluated. In addition, as ECA rules that form complex topologies can increase the response time, FORK-type and JOIN-type topologies are evaluated in Section 5.3 to confirm if requirement 6 is satisfied. All the results in these sections are the averages of five trials.

5.1 Resource Consumption

As a basic performance evaluation, the resource consumption was measured for different phases of rule engine execution under normal status, event evaluation, and action execution. OCCUR tag and LOGWRITE tag are utilized as the event and action, because they are the most common tags. Normal status means the phase before event evaluation; the rule engine is idle. By comparing the result of the normal phase with its counterpart under event evaluation and action execution, the impact of the rule engine can be determined.

As shown in Figure 7, CPU load for both event evaluation and action execution was less than 5%. In particular, the CPU load for event evaluation was almost the same as under normal status. This means that OCCUR tag evaluation generates negligible CPU load, and the rule engine doesn't negatively impact system operation or other applications.

The RAM consumption was about 300 KB throughout all phases of rule engine execution. We consider this result to be sufficiently lightweight for a smartphone with 512 MB of RAM. It is not considered likely that RAM consumption will be a problem even if the evaluation scenario is complicated.

5.2 Response Time

The response time of the rule engine deteriorates when the event is evaluated against certain time span of terminal log. Because SUM tag and SUB tag correspond to this type of ECA tag, we evaluated their response times.

The processing times of SUM tag for target time spans from 15 to 120 minutes show that the processing time is less than 100 ms regardless of the time span (Figure 8).

The equivalent processing time of SUB tag is less than 50 ms (Figure 9). Though the processing time deteriorates with time span duration, it still satisfies requirement 6.

5.3 Response Time of Various Topologies

We evaluated the impact of context processing pattern on response time. Several ECA rules with JOIN type and FORK type topologies were assessed.

The processing time of the JOIN-type topology showed about the same result for different numbers of executed events; all results were under 100 ms (Figure 10).

The processing time of FORK-type topology increases with the number of executed actions. With four and eight actions, the processing time of event evaluation was 79 ms and 305 ms, respectively (Figure 11).

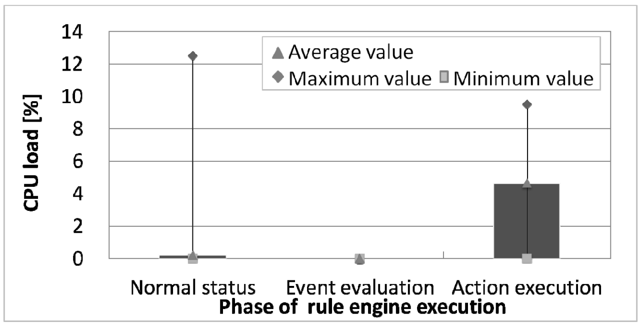


Figure 7: CPU load of rule engine execution

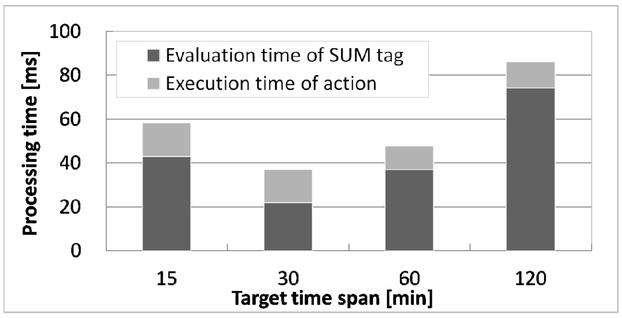


Figure 8: Processing time of SUM tag

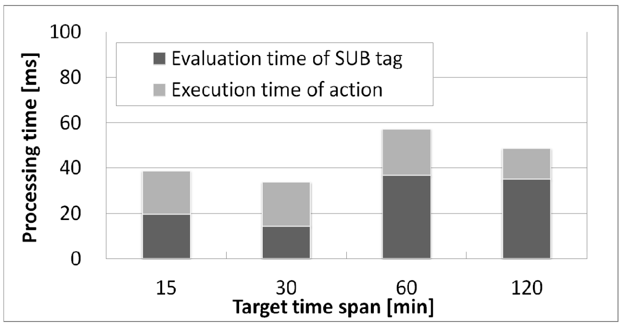


Figure 9: Processing time of SUB tag

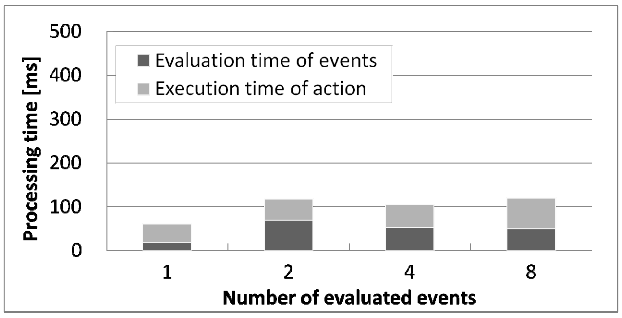


Figure 10: Processing time of JOIN-type topology

From these results, the processing time of the JOIN-type topology fully complies with requirement 6. The FORK-type topology is also practical since the number of actions trig

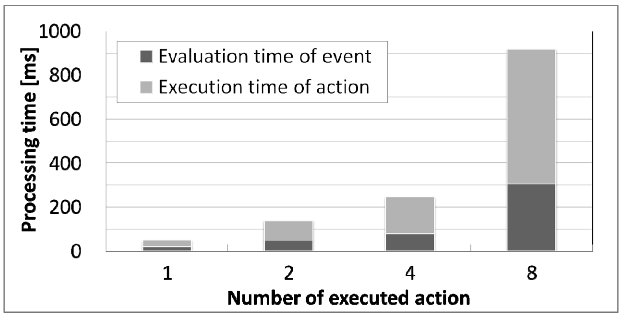


Figure 11: Processing time of FORK-type topology

gered by a certain user context is considered to be around three at most.

6 CONCLUSION

We have proposed an application execution environment that allows developers of context aware applications to utilize various terminal logs, so that user context can be flexibly leveraged. We tackled the challenge of allowing the application developers to easily create algorithms employing user context detection. To this end, we proposed an ECA rule specification that satisfies the functional requirements of covering the five semantic dimensions, *W4H*, handling various data collected from sensor devices and smartphone middleware, and supporting composite contexts.

Extensive evaluations of a prototype on a commercial smartphone confirmed that rules can be processed under 100 ms for SUM tag and SUB tag for single ECA rules, and under 305 ms for more complex FORK-type and JOIN-type topologies. Overall, the evaluation results satisfied the requirements set. Future work includes realizing a user-friendly interface to support the development and customization of context-aware applications.

REFERENCES

[1] P. Moore, B. Hu, and M. Jackson, Rule Strategies for Intelligent Context-Aware Systems, IEEE CISIS (2011).

[2] D. Kulkarni and A. Tripathi, A Framework for Programming Robust Context-Aware Applications, IEEE Trans. Software engineering, Vol.36, No.2, pp.184-197 (2010).

[3] P. Moore, M. Jackson, and B. Hu, Constraint Satisfaction in Intelligent Context-Aware Systems, IEEE CISIS (2010).

[4] G. Pallapa, N. Roy, and S. Das, Precision: Privacy Enhanced Context-Aware Information Fusion in Ubiquitous Healthcare, IEEE SEPCASE, (2007).

[5] Y. Chon and H. Cha, LifeMap: A Smartphone-Based Context Provider for Location-Based Services, IEEE Pervasive Computing, (2011).

[6]A. M. Khan, Y. K. Lee, S. Y. Lee, and T. S. Kim, Human Activitiy Recognition via An Accelerometer-Enabled-Smartphone Using Kernel Discriminant Analysis, IEEE FutureTech, (2010).

[7]A. Bujari, B. Licar, and C. E. Palazzi, Road Crossing Recognition through Smartphone's Accelerometer, IFIP Wireless Days, (2011).

[8] R. Guo, T. Zhu, Y. Wang, and X. Xu, MobileSens: A Framework of Behavior Logger on Android Mobile Device, IEEE ICPCA, (2011).

[9] P. D. Costa, J. P. Almeida, L. F. Pires, and M. Sinderen, Evaluation of a Rule-Based Approach for Context-Aware Services, IEEE GLOBECOM (2008).

[10]T. Hu, and B.Li, Research on the Mechanism of Tourism Information Change Management Based on ECA Rules, IEEE CSE, (2010).

[11] Jari Forstadius, Ora Lassila, and Tapio Seppanen, RDF-based Model for Context-aware Reasoning in Rich Service Environment, IEEE PerCom Workshops, (2005).

[12] Daniel Salber, Anind K. Dey and Gregory D. Abowd, The Context Toolkit: Aiding the Development of Context-Enabled Applications, ACM CHI, (1999).

[13] Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Keranen, and Esko-Juhani Malm, Managing Context Information in Mobile Devices, IEEE Pervasive Computing, (2003).

[14] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, MyExperience: A System for *In situ* Tracing and Capturing of User Feedback on Mobile Phones, ACM 5th international conference on Mobile systems, (2007).

[15] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, P. Kortum, LiveLab: Measuring Wireless Networks and Smartphone Users in the Field, ACM SIGMETRICS Performance Evaluation Review, (2011).

[16] R. F. B. Neto and M. G. C. Pimentel, Toward a Domain-Independent Semantic Model for Context-Aware Computing, Third Latin American Web Congress, (2005).

[17] W. Na, S. Cho, E. Kim, and Y. Choi, Event Detection in Composite Context Aware-Service, IEEE ICUFN, (2011).

[18] S. Kim, E. Kim, and Y. Choi, Composite Context Information Design and Model Approach for Adaptive Service Detection, IEEE 13th ICACT, (2011).

[19] E. Gultawatvichai and T. Senivongse, A Development of Process-Based Composite Contexts for Mobile Device Platforms Based on Model Driven Architecture, IEEE JCSSE, (2011).