sustainability

MDPI

Article

Exclusive Contexts Resolver: A Low-Power Sensing Management System for Sustainable Context-Awareness in Exclusive Contexts

Dusan Baek and Jung-Won Lee *

Department of Electrical and Computer Engineering, Ajou University, 206, Suwon 16499, Korea; darkdusan@ajou.ac.kr

* Correspondence: jungwony@ajou.ac.kr; Tel.: +82-31-219-1813

Academic Editor: James J. Park
Received: 3 March 2017; Accepted: 17 April 2017; Published: 19 April 2017

Abstract: Several studies focus on sustainable context-awareness of a mobile device to which power is supplied from a limited battery. However, the existing studies did not consider an unnecessary sensing operation in exclusive contexts wherein it is not possible for the exclusive contexts to logically exist at the same time and are instead occasionally inferred practically due to the inaccuracy of the context-awareness. Simultaneously inferring two or more exclusive contexts is semantically meaningless and leads to inefficient power consumption, and thus, it is necessary to handle this problem for sustainable context-awareness. To this end, in the present study, an exclusive contexts resolver (ExCore), which is a low-power sensing management system, is proposed for sustainable context-awareness in exclusive contexts. The ExCore takes the sensor behavior model to the developer and identifies the sensing operation inferring the exclusive contexts through unnecessary sensing operation search rules. It also generates low-power sensing operations. The application and middleware were evaluated with the low-power sensing operations generated by the ExCore. The results indicated an average power efficiency improvement of 12–62% depending on the test scenario. The ExCore helps application developers or middleware developers in providing sustainable context-aware service in exclusive contexts.

Keywords: low-power; context-aware; sustainable; exclusive contexts; sensor management; sensing behavior model; context-sensing model

1. Introduction

The availability of a mobile device equipped with various sensors allows users to utilize context-aware services [1,2]. The context-aware service senses the physical environment around the user by using various sensors to offer a well-fitted service. However, the context-aware service requires continuous sensing to monitor a user's context, and thus, it consumes a high amount of power. Specifically, the limited power supply from the battery becomes an important issue for the mobile device in question [3].

Hence, several studies have focused on the sustainable context-awareness of mobile devices. Generally, there are two kinds of solutions for sustainable context-awareness: application-independent solutions and application-dependent solutions. However, the application-independent solutions have limitations in solving power-related problems in mobile devices due to the over consumption of resources by the faulty design of applications [4]. If the system indiscriminately blocks application commands to preserve power efficiency, this causes a direct drop in QoS (Quality of Service). Conversely, if the system accepts a request for an unnecessary operation due to a simple design error, this results in inefficient power consumption. Thus, application-dependent solutions are required and

ongoing research focuses on low power to solve these problems. This research aids researchers or developers in localizing the energy bug [5] (that is also termed the energy black hole [6,7] or energy leak [8]), which causes inefficient power consumption at the application level (component level or instruction level in detail). This information can be used by developers to fix the energy bugs and improve power efficiency.

Despite the contributions of extant research, mobile device users continue to suffer from limited battery power. As previously mentioned, context-awareness consumes additional power, and thus power efficiency is an extremely important issue. Thus, a few studies focused on sustainable context-awareness and proposed additional low-power methods. Specifically, they used the similarity of contexts or context hierarchy information to prevent the occurrence of identical sensing operations or to perform sensing operations that are maximally efficient among the sensing operations that enable the same context [9,10]. That is, in the case of a context-aware application, an additional low-power option can be achieved by using a context-specific low-power scheme in addition to a traditional low-power scheme.

However, in spite of these efforts, the previous studies had limitations because they do not consider an unnecessary sensing operation in exclusive contexts wherein it is not possible for the contexts to exist logically at the same time. For example, in an indoor/outdoor context-aware service, indoor and outdoor contexts can be inferred when a Wi-Fi signature is acquired and the GPS (Global Positioning System) signal is received, respectively. It is not possible for indoor and outdoor contexts to coexist logically at the same time. However, the afore-mentioned contexts are often inferred by Wi-Fi and GPS sensors simultaneously (for example, at an edge of a building). In a manner similar to this case, it is semantically meaningless to infer two or more exclusive contexts simultaneously, and this leads to inefficient power consumption. Hence, it is necessary to handle the problem for sustainable context-awareness. Figure 1 shows the context classified by an operational categorization used in a previous study [11] to illustrate the practical reasons for the occurrence of exclusive contexts.

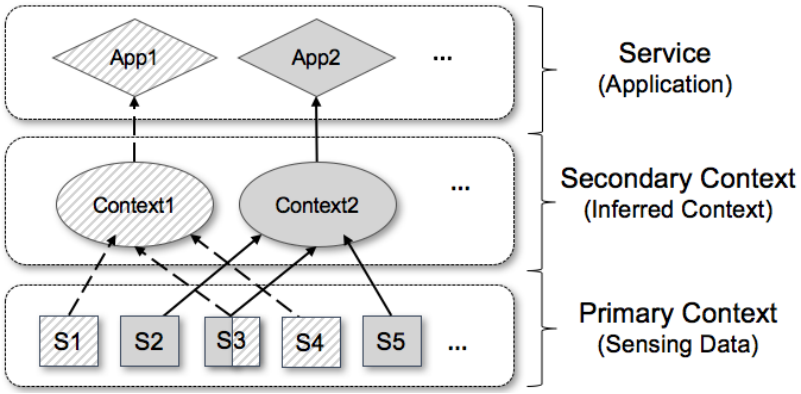

Figure 1. Context classification by the operational categorization.

As shown in Figure 1, the primary context corresponds to the raw value obtained from actual sensing operations such as GPS location fix and acceleration value. This is utilized to inform secondary contexts based on the business logic of a context-aware application. In conventional methods for low-power consumption, low-power can be achieved by caching the secondary context. Conversely, the inference of the exclusive contexts occurs in the primary context. With respect to the assumption that context 1 and context 2 in Figure 1 are exclusive contexts, sensing data values, such as (S1 and S2), (S4 and S5), or (S2 and S4), which generate these secondary contexts will not be obtained at the same time. However, in a realistic scenario, the afore-mentioned sensing data values, such as (S3 and S4) or (S1 and S3), are occasionally sensed simultaneously. This is because the accuracy of a context-aware service does not perfectly match the primary context and the secondary context.

As a result, the power consumed for inference is unnecessary because the inferred exclusive contexts are not logically meaningful. Therefore, it is necessary to reduce unnecessary power consumption in exclusive contexts for sustainable context-aware service, and this entails the following process.

First, it is necessary to identify the exclusive contexts and the unnecessary sensing operation. In this study, it is assumed that target services are scalable and that their developers could be different. Hence, a context-sensing model is proposed to express the context and sensing operation in each service. The context-sensing model is used to identify unnecessary sensing operations in exclusive contexts and to suggest the means for an efficient operation. The context-sensing model is based on the ease of use and extensibility, and therefore, it is lightweight when compared with the existing sensing models based on interoperability, robustness, and self-adaptability. In particular, all necessary information can be extracted from the automata for the sensing behavior model that is basically generated in the development process, and thus, this does not require the additional effort of a developer representing the context-sensing model. The system proposed in this study (that is, the Exclusive Contexts Resolver; ExCore) can automatically perform a translation from automata to the context-sensing model. The context-sensing model is utilized as a fact in the expert system in ExCore, and it identifies unnecessary sensing operations in the exclusive contexts.

Second, it is necessary to efficiently control the inefficient sensing operation in exclusive contexts for a sustainable context-aware service. To this end, the ExCore provides an environment to suggest and retrieve an efficient sensing operation method. The rules of identifying the inefficient sensing operation in the exclusive contexts as well as dealing with the same can be applied to the expert system in the ExCore. In addition, efficient sensing operation methods in exclusive contexts are shared by a large number of developers, and thus, there is room for improvements in terms of the power efficiency. As a result, the proposed system helps application developers or middleware developers in providing sustainable context-aware services with respect to exclusive contexts.

In this study, the ExCore presented is applied to a scenario in which exclusive contexts occur. The results indicate an improvement of 12%–62% in the power efficiency of the exclusive contexts. It is expected that the ExCore will aid in more efficient power consumption in future heterogeneous situations, as the number of context-aware applications is expected to increase in the future.

The remainder of this study is organized as follows. Section 2 reviews the literature related to the models and methods for sustainable context-awareness. Section 3 addresses the exclusive contexts and the reasons for the related existing problems. Section 4 proposes the ExCore that identifies the inefficient sensing operation in exclusive contexts and provides an efficient sensing operation method. Section 5 evaluates the ExCore by applying an efficient sensing operation for the application and the middleware in Android OS. Finally, Section 6 presents the conclusions and directions for future research.

2. Related Work

2.1. Models for Sustainable Context-Awareness

The representation of context-awareness is classified into general models for multiple applications, domain-specific models for specific applications, and no-models that directly use sensing data based on a classification in a previous study [12,13]. In order to manage and operate multiple applications for sustainable context-aware service, it is essential to express their context-aware services by using general models. There are various types of general models as described below [13]:

① Key-value models represent the simplest data structure for modeling contexts by exploiting pairs of two items, namely a key (attribute name) and its value.
② Markup scheme models use XML-based representations to model a hierarchical data structure consisting of markup tags, attributes, and contents.
③ Object-oriented models use the benefits of the object-oriented approach, namely encapsulation and reusability, and each class defines a new context type with associated access functionalities.

④ Logic-based models use the high expressiveness intrinsic to the logic formalism, and the context contains facts, expressions, and rules, while new knowledge can be derived by inference.

⑤ Ontology-based models use ontologies to represent context and utilize the expression capability related to even complex relationships, and data validity is typically expressed by imposing ontology constraints.

It is necessary for the criteria to select the most suitable models for sustainable context-awareness among the various types of the models. This entails the following: First, context-aware services have their own distinct business logic. Therefore, it is not possible to pre-define all the context-related information nor it is possible to enforce selection from several pre-defined models. Additionally, context-aware services emphasize creativity and independence as opposed to interoperability. Therefore, it is not possible to predefine the model through technology such as ontology. It is necessary to provide a reference model such that developers can express their own situation and sensor information. Second, the model should be expressed at a level that does not overburden the developer. Currently, context-aware applications are developed without considering the integration of services. Therefore, additional information required by the afore-mentioned model could pose a burden to the developer, and thus these types of activities do not benefit developers.

Given the above reasons, studies on sustainable context-awareness define their individualized models to represent services, contexts, and sensor information. A previous study [9] used an XML-based model to define sensor state and transition. This method is simple and can reduce the burden on the developer and has the added advantage that it can be extended to various services, situations, and sensing operation methods. An extant study [6] defined a logic-based model, that is, an application execution model to analyze sensory data utilization. This model enables rule-based reasoning, and it is possible to apply information flow tracking through tainting propagation. However, it is difficult to apply this model to a context-aware service because it is based on a sensor level. Another study [10] used a logic-based model to display sensors and contexts. Logic-based models have the advantage of reasoning as well as the disadvantage that it is necessary for a developer to separate efforts to generate the models as specified in an extant study [6].

2.2. Low-Power Methods for Sustainable Context-Awareness

Studies for sustainable context-awareness can be divided into studies for low-power methods of general mobile services and studies of context-aware services. First, mobile manufacturers or OS vendors work towards improving the power efficiency of mobile devices. Mobile manufacturers developed a 'big-little architecture', which selectively uses high-performance and low-end tasks [14]. Additionally, they proposed a 'sensor hub' that processes sensor events with a processor that consumes less power when compared with that of a CPU [15]. The application of these methods allows the achievement of power efficiency at the system level. Recently, Google, which is an OS vendor, has adopted new technology to reduce power consumption whenever a new version of Android is released at the OS level. An example involves 'batching' that processes sensing operations through a processor that consumes less power and periodically processes the sensor data in a batch to reduce the power of the application processor (AP). Another example is that of the 'Doze mode' that reduces power consumption by delaying CPU and network activity when not in charge and when switched off. Additionally, 'Project svelte' reduces the reception power of broadcasts by eliminating nonessential implicit broadcasts [16].

Furthermore, studies related to low-power consumption are also underway at the application-level. They can be classified into studies that identify the causes of inefficient power consumption and studies that remove the causes. The goal of the former studies includes aiding developers to localize the energy bug (also called energy black hole or energy leak) that causes inefficient power consumption [17–19]. The afore-mentioned studies involved creating a power model at the component level [17,18] or at the instruction level [19] and estimating and analyzing the power consumption of the context-aware service by using the model. However, these studies have a

disadvantage as they only provide information regarding the amount of the power consumption without any additional information. An extant study [6] proposed an approach to remedy the shortcoming of the previous studies by systematically diagnosing energy inefficiency problems. This approach provides information related to the misusage of the sensor listener and underutilization of the sensory data by analyzing the sensor utilization. The goal of the latter studies involves aiding a developer in fixing the localized energy bugs. An extant study [20] presents APE, which is an annotation language and middleware service that eases the development of an energy-efficient Android application. In this manner, a developer can identify and improve the energy bug to enable efficient power consumption.

In addition to low-power studies for general mobile services, low-power studies are also conducted for context-aware services. They focus on the context of context-aware applications and use the same for efficient sensing operations. An extant study [3] presented a rate-adaptive positioning system that adjusts the periodic duty-cycle of GPS based on required accuracy. A study [9] examined a design framework for sensor management to substitute existing sensors with a minimum set of sensors that consumes lower power. Another study [10] proposed an ACE (Acquisitional Context Engine), which is a middleware that exploits the sensing data cache or infers a context from a previously-known context without redundant additional sensing. These studies provide solutions to address inefficient power consumption. However, the studies do not consider the exclusive contexts. Therefore, it is hard to establish efficient power consumption with respect to the exclusive contexts. In order to provide the sustainable context-awareness in consideration of the exclusive contexts, we proposed a low-power sensing management method in a previous study [21]. However, the study only focused on the possibility to reduce the power consumption in the exclusive contexts. Therefore, the responsibility of identifying unnecessary sensing operation and remedying the operation is assigned to the developer. On the other hand, ExCore could identify the unnecessary sensing operations according to the developer's context and generate the efficient sensor operating instruction automatically.

3. Exclusive Contexts

3.1. Problem Definition

This subsection discusses the problem definition to aid in understanding the meaning of exclusive contexts and the reason as to why exclusive contexts cause inefficient power consumption by describing a scenario. The scenario includes the following:

- Scenario.

Tom, a graduate student, usually takes the car to the lab. He studies in the lab and attends a seminar with a professor. He utilizes a context-aware service that provides three key functions. First, the service blocks calls while Tom drives and sends messages informing callers about Tom's driving state. Additionally, the service changes the ringer to the vibrating mode when Tom studies and to the mute mode when Tom attends seminars. The service classified context into the following three types: The driving context (Situation A) if the GPS sensor receives satellite data. The studying context (Situation B) if the Wi-Fi signal of the office/laboratory can be received and sound around the device exceeds a pre-defined threshold. The seminar context (Situation C) if the Wi-Fi signal of the office/laboratory can be received and sound around the device is less than the pre-defined threshold.

The driving, studying, and seminar contexts correspond to the exclusive contexts that cannot logically coexist at the same time. However, if the GPS sensor receives satellite data and the Wi-Fi sensor can receive the Wi-Fi signal of the office/laboratory, then the driving and studying context or (Situation D) the driving and seminar context is inferred. That is, the simultaneous inference of two or more exclusive contexts is semantically meaningless, and thus the sensing activity inferring these contexts is unnecessary. Therefore, it is necessary to manage the sensing to infer these types of meaningless contexts for the development of sustainable context awareness.

3.2. Efficient Sensing in Exclusive Contexts

Figure 2 describes the problem definition based on the classification by the operational categorization as given in a previous study [19].

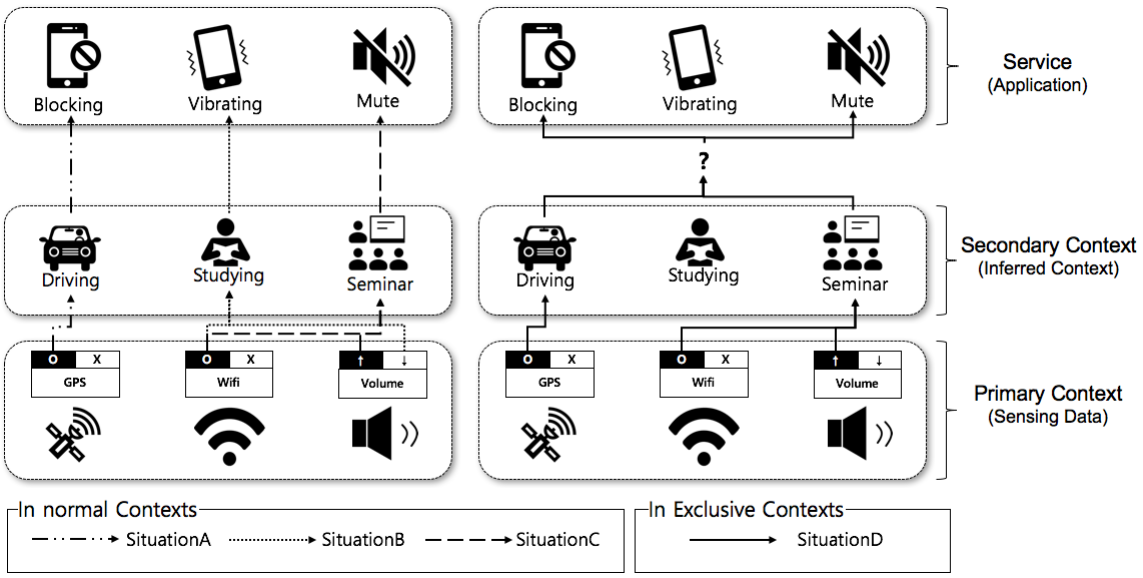

Figure 2. The context classification of problem definition by the operational categorization.

As shown in the Figure 2, each context-aware application provides a service based on the context of the user. Additionally, each context is inferred by analyzing the sensing data of the sensors in the device. Therefore, the developer of the context-aware service sets the provision of context appropriate to the service and then commands the sensing operation to infer the context. The sensing data collected through the sensing operation is used to confirm whether or not the data supports the context. The context inference algorithm used at this time is designed and implemented by the developer. Therefore, if the accuracy of the inference algorithm is 100%, then exclusive contexts that cannot logically exist at the same time will not be inferred simultaneously such as the driving and seminar, driving and studying, and studying and seminar contexts. However, if the accuracy of the inference algorithm is not 100%, then it is possible to infer exclusive contexts due to incorrect inference of the context. If such exclusive contexts occur simultaneously, then it is not possible to provide a well fitted-service based on the user's context, and therefore the power consumed in this type of a situation becomes meaningless. This study involves accounting for the exclusive contexts that are not considered by existing context-aware services and reduces inefficient sensing operations in exclusive contexts for sustainable context-awareness. In other words, in this paper, we try to improve the efficiency of power consumption by correcting the inefficient power consumption caused at the application level (faulty design, etc.) through the context-related information.

Even if the context inference algorithm is not perfect, all the exclusive contexts do not actually occur. In the example shown in Figure 2, the studying context and the seminar context do not occur simultaneously irrespective of the accuracy of the context inference algorithm. This is because sensing data that supports the exclusive contexts are disjointed at the primary context level. When no Wi-Fi signal is received, both contexts are not inferred. When a Wi-Fi signal is received, only one of the two contexts will be inferred according to the condition based on the pre-defined volume threshold. It is not possible for the data (events) collected by a sensor to have more than two values simultaneously, and therefore the contexts cannot be simultaneously inferred if the types of sensors used to infer exclusive contexts and their values are disjointed. On the other hand, in the case that the sensors or their values are not disjointed, exclusive contexts could be inferred. In the example shown in Figure 2,

the driving context uses a GPS, and the studying context and seminar context are inferred using a Wi-Fi and a volume sensor. Therefore, exclusive contexts, such as Driving context and Studying Context and Driving Context and Seminar context, can be inferred by a GPS, Wi-Fi, and a volume sensor. Therefore, in order to enable sustainable context-awareness in exclusive contexts, it is necessary to consider contexts in exclusive contexts as well as the types and conditions of the inferring sensors.

Currently, exclusive contexts are not often inferred, and thus there are limited opportunities for generating low power through these contexts. However, upcoming IoT environments will involve the mixing of heterogeneous devices, context-aware services, and sensors, and thus exclusive contexts will occur frequently. The present study utilizes an expert system to present solutions in this complex situation. An expert system is a system developed to provide a software system with the same intellectual abilities as an original expert. It can effectively grasp complex facts and the logical relationship between the facts. In this study, an expert system is used to identify the exclusive contexts, and an efficient sensing operation is proposed.

4. ExCore: Exclusive Contexts Resolver

4.1. ExCore Overview (System Overview)

Figure 3 describes an overview of the proposed system used in this study, which is termed as the 'Exclusive Contexts Resolver' and abbreviated as ExCore.

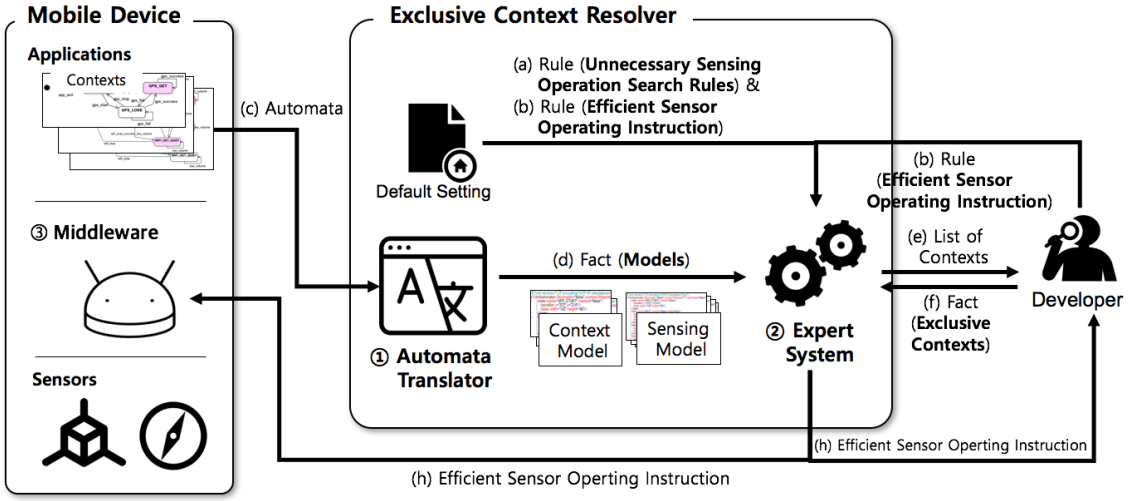

Figure 3. An overview of the ExCore system.

4.1.1. Automata Translator

It is necessary to analyze context-related information used in service and sensing-related information to infer the context, in order to eliminate unnecessary power dissipation in exclusive contexts. As previously mentioned, various modeling methods exist to define context and sensing-related information. Most of the methods require very detailed and versatile information for operations including interoperability, scalability, and self-adaptability, and this places a burden on developers. In this study, the system only takes automata (sensing behavior model, c) as an input to avoid this burden. However, it is not possible for the expert system to directly read automata, and therefore it is necessary to convert the automata to a form that can be used in an expert system. The Automata translator translates the automata into a context-sensing model (f). The ExCore inputs this context-sensing model to the expert system as facts, and the expert system deduces the unnecessary sensing operations related to the exclusive contexts.

4.1.2. Expert System

The expert system analyzes the context and sensing-related information as facts and identifies the sensing operation that consumes unnecessary power within exclusive contexts by using pre-defined unnecessary sensing operation search rules as defaults (a). This is followed by presenting an efficient sensor operating instruction (h) to enable sustainable context-awareness. The context of the sensing behavior model added by the developer corresponds to a natural language, and the meaning differs based on each individual developer. Hence, ExCore is fed back (e,f) by querying the developer as to whether or not the context to be added to the developer and the existing context are exclusive contexts. Additionally, the developer can propose a more efficient sensing operation by adding an efficient sensor operating instruction (b) as a rule. The efficient sensor operating instruction deduced by the expert system is delivered to the developer. The developer utilizes this instruction to develop an application or a middleware to enable an efficient sensing operation with respect to the exclusive contexts. The sensing operation on the middleware enables efficient power consumption without requiring additional changes in the application code. In order to perform this, it is necessary for the middleware to possess the ability to control various sensors by following the instruction.

4.2. Context-Sensing Model

In order to improve the power efficiency in the exclusive context, it is necessary to analyze context-related information used in the service and sensing-related information used to infer the service. The extent of information analysis and the manner in which the analysis is handled are highly dependent on the modeling of the information. The requirements for the modeling in the proposed system include the following:

- Adaptability and Scalability: A model should be provided to developers such that they can express their own context and sensor-related information.
- Simplicity: Only minimal information should be required for sustainable contexts such that developers are not burdened.

ExCore only requires automata (sensing behavior model) with respect to the developer to satisfy the requirements. It is assumed that the automata correspond to a minimal artifact produced during the development of a context-aware service. However, if many sensors are used and describing all of the sensors is difficult, the developer is required to make additional efforts in selecting and describing only the sensors and their operations corresponding to the contexts that cause exclusive contexts. Figure 4 shows an example of the automata that expresses the sensing behavior to infer the seminar context of the Problem Definition as illustrated in Section 3.1.

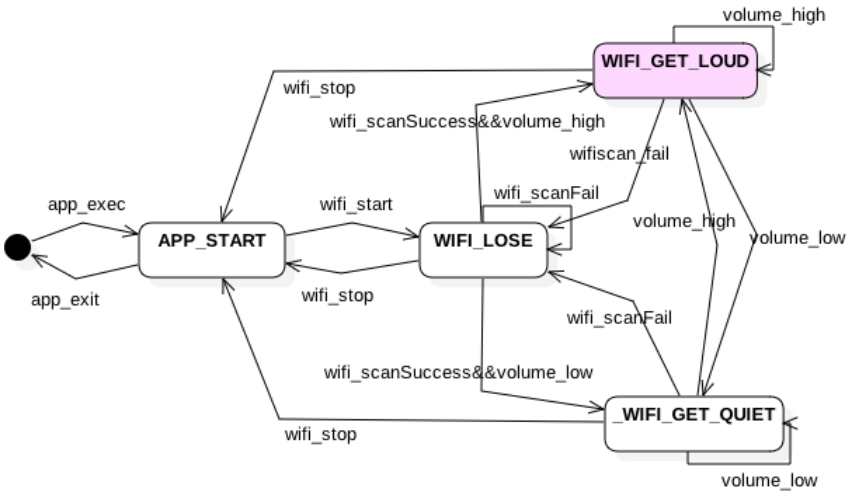

Figure 4. The sensing behavior model for context-awareness of the seminar context.

The automata is an event driven model that consists of states and transitions and it represents the sensing behavior of sensors for context-awareness. Each state is maintained when it receives sensing data that does not change beyond the threshold, and the service infers the context based on the corresponding state. If sensing data with a change that exceeds the threshold value is received as a result of the periodic sensing operation, then it transits to another state based on the condition of the transition. Each automata is described by each context used to trigger the service, and this is termed as the core context (that is, if there are three contexts, then it is necessary to create three automata). Therefore, the automata consists of a core context and other contexts that are termed as 'support contexts'. The state inferring the core context is denoted as the marked state. The marked state is used to identify the exclusive contexts.

ExCore receives the sensing behavior model of the automata type that infers individual contexts and inputs the same as a fact in the expert system. The sensing behavior model satisfied the requirements for the modeling as previously mentioned. However, this is not directly usable because it is too complex to use in the expert system. Therefore, it is necessary to change the sensing behavior model from the expert system. The Automata Translator is responsible for these tasks, and as a result, it generates the context-sensing model. Figure 5 illustrates the context-sensing model proposed in this study.

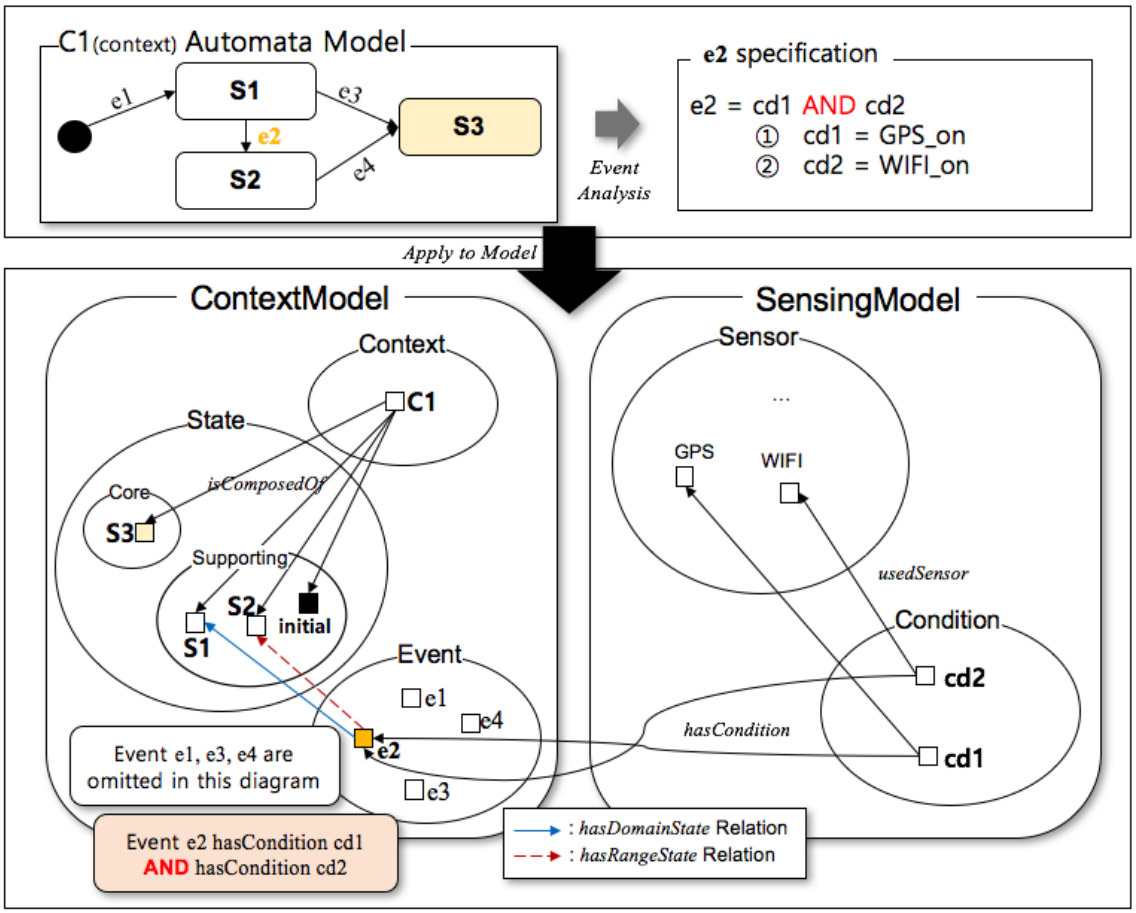

Figure 5. The sensing behavior model and the context-sensing model.

The bottom of Figure 5 illustrates a context-sensing model with the goal of expressing the values of the sensing behavior model in the expert system. The context-sensing model consists of a context-model that contains abstract information related to the context and a sensing model that is responsible for explicit information related to the actual sensing data. Additionally, they are composed

of entity-related information and relation-related information and are designed to be easy to interpret by the expert system. In detail, the context of a context-sensing model is composed of states, and the states are divided into marked states deducing the core context and supporting states deducing the support contexts. Each state is transited by the event of the sensors and has a domain state that corresponds to the start of transition and a range state that corresponds to the arrival of the transition. The event occurs when one or more sensors (GPS, Wi-Fi, etc.) satisfy the condition. Figure 6 shows a context-sensing model of the sensing behavior model generated from the seminar context in Section 3.1.

- **Context**
  - {Seminar}
- **State**
  - [{APP_START, supporting}, {WIFI_LOSE, supporting}, {_WIFI_GET_QUIET, core}, …}
- **Event**
  - { app_exec, app_exit, wifi_start, wifi_stop, wifi_scanFail, wifi_scanSuccess&&volume_high, …}
- **Condition**
  - {exec, exit, stop, start, scanSuccess, scanFail, low, high}
- **Sensor**
  - {app, wifi, volume}
- **isInferredBy**
  - [{Seminar, APP_START}, {Seminar, WIFI_LOSE}, {Seminar, WIFI_GET_LOUD}, …]
- **hasDomainState**
  - [{app_exec, IDLE}, {app_exit, APP_START}, {wifi_start, APP_START}, {wifi_stop, WIFI_LOSE}, …
- **hasRangeState**
  - [{app_exec,APP_START}, {app_exit, IDLE}, {wifi_start,WIFI_LOSE} , {wifi_stop,APP_START}, …
- **hasCondition**
  - [{app_exec, app, exec}, {app_exit, app, exit}, {wifi_start, wifi, start}, {wifi_stop, wifi, stop}, …

Figure 6. The context-sensing model of the seminar context in Section 3.1.

In Figure 6, some events are composed of two more sensing conditions such as *wifiscan_success*&&*vlume_high*. The context-sensing model has a way to represent the AND and OR transitions, and Figure 7 is an example that illustrates the basic, AND, and OR transitions.

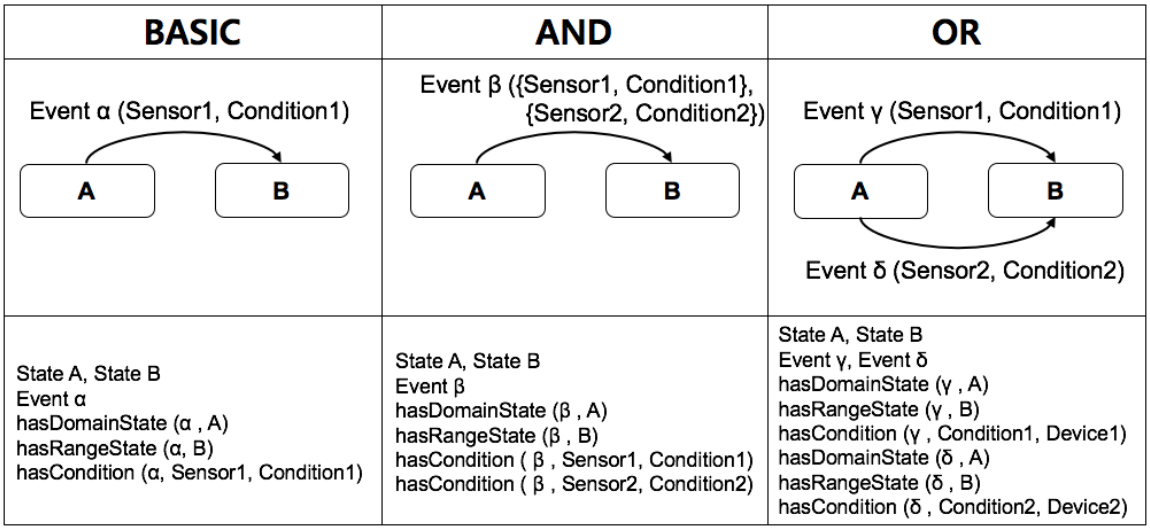

Figure 7. The context-sensing model of basic, AND, and OR transitions.

In basic transition case in Figure 7, when the Sensor1 satisfies *Condition1*, then *Eventα* is triggered, and it makes a transition from the *StateA* to the *StateB*. In the AND transition case, the *Eventβ* could be triggered when both the *Sensor1* and *Sensor2* satisfy the *Condition1* and *Condition2* at the same time, respectively. For adapting our model to the expert system, we should use only the static number

of tuples for the model. For this reason, we separate one *hasCondition* into two *hasConditions* for representing the AND transition. In the OR transition case, a transition from the *StateA* to the *StateB* is triggered when either *Sensor1* satisfies *Condition1* or *Sensor2* satisfies *Condition2*. Therefore, we separate the event making transition into two events for each sensing trigger, such as the *Event*γ and the *Event*δ. By using a combination of these transition expressions, we can represent all kinds of sensing behavior models as the context-sensing model.

4.3. Low-Power Sensing

The Context-Sensing model generated by the Automata Translator is added as facts to the expert system. Figure 8 shows the templates for the facts added to Jess, which is one of the famous expert systems and one example among the facts of the seminar context in Section 3.1 according to the templates.

- **Context**
  - (deftemplate MAIN::context extends MAIN::__fact (slot id) (slot name))
  - (MAIN::context (id User_context_0) (name Seminar))
- **State**
  - (deftemplate MAIN::state extends MAIN::__fact (slot id) (slot name) (slot isItCore))
  - (MAIN::state (id User_state_4) (name _WIFI_GET_QUIET) (isItCore core))
- **Event**
  - (deftemplate MAIN::event extends MAIN::__fact (slot id) (slot name))
  - (MAIN::event (id User_event_0) (name app_exec))
- **Condition**
  - (deftemplate MAIN::condition extends MAIN::__fact (slot id) (slot name))
  - (MAIN::condition (id undefined) (name exec))
- **Sensor**
  - (deftemplate MAIN::sensor extends MAIN::__fact (slot id) (slot name))
  - (MAIN::sensor (id User_Sensor_0) (name app))
- **isInferredBy**
  - (deftemplate MAIN::isInferredBy extends MAIN::__fact (slot contextId) (slot stateId))
  - (MAIN::isInferredBy (contextId User_context_0) (stateId User_state_0))
- **hasDomainState**
  - (deftemplate MAIN::hasDomainState extends MAIN::__fact (slot eventId) (slot stateId))
  - (MAIN::hasDomainState (eventId User_event_0)) (stateId User_state_0)
- **hasRangeState**
  - (deftemplate MAIN::hasRangeState extends MAIN::__fact (slot eventId) (slot stateId))
  - (MAIN::hasRangeState (eventId User_event_0)) (stateId User_state_1)
- **hasCondition**
  - (deftemplate MAIN::hasCondition extends MAIN::__fact (slot eventId) (slot sensorId) (slot conditionId))
  - (MAIN::hasCondition (eventId User_event_0) (sensorId User_Sensor_0) (conditionId User_condition_0))

Figure 8. The templates for the facts and one example among the facts according to the templates.

All name of entities and relations are named by a developer using natural language. To identify each element of the context-sensing models, we add an *id* slot to all entities, and all relations could refer the entities only using this *id*.

When the facts for a new context are added, the expert system queries the developer as to whether or not the new context makes new exclusive contexts with the contexts which have been stored as the facts in the expert system. If it is part of the exclusive contexts, it is also added as a fact. Then, the expert system deducts the sensing operation that consumes unnecessary power using the unnecessary sensing search rule defined in this paper. Here are the necessary and sufficient conditions for the unnecessary sensing operation:

① There are no sensors of the same type used for the exclusive contexts, and the exclusive contexts are inferred at the same time.

② If the conditions of the sensors that have the same type and which are used for the exclusive contexts are not same, they are not disjointed. Also, the exclusive contexts are inferred at the same time.

First of all, for identifying the unnecessary sensing operation, we should check whether there are any sensors of the same type used for the exclusive contexts. Rule 1 is responsible for this as shown in

Figure 9. After Rule 1 is fired by the facts in the expert system, the latter of the unnecessary sensing operation ② would be checked. Otherwise, the former ① would be checked.

Rule.1 usedSameSensor search Rule

```
1:  ∃x1 ∃x2 ∃x3 ∃x4 ∃x5 ∃y1 ∃y2 ∃y3 ∃y4 ∃y5
2:  (context (id x1)) ∧ (context (id y1))
3:  ∧ (areExclusiveContexts (contextId x1) (contextId y1))
4:  ∧ (not (equal (x1) (y1)))
5:  ∧ (isInferredBy (contextId x1) (stateId x2))
6:  ∧ (isInferredBy (contextId y1) (stateId y2))
7:  ∧ (state (id x2) (isItCore "core"))
8:  ∧ (state (id y2) (isItCore "core"))
9:  ∧ (hasRangeState (eventId x3) (stateId x2))
10: ∧ (hasRangeState (eventId y3) (stateId y2))
11: ∧ (hasCondition (eventId x3) (sensorId x4) (conditionId x5))
12: ∧ (hasCondition (eventId y3) (sensorId y4) (conditionId y5))
13: ∧ (equal (x4) (y4))
    ⇒
14: (usedSameSensors (contextId x1) (eventId x3) (contextId y1) (eventId y3))
```

Figure 9. Rule1: The rule for checking whether the same sensors are used.

The unnecessary sensing operation could be divided into two types based on the result of Rule 1, and each of them is discerned by applying different forward chaining rules. First, Rule 2 finds the unnecessary sensing operation when there are no sensors of the same type in Figure 10. It checks whether there exists a fact satisfying the conditions (2–13) among the facts in the expert system by the existential quantifier (1). If there is an unnecessary sensing operation, the fact (14) is added in the expert system. To examine this rule closely, it checks whether there are two different contexts (2, 4) which are the exclusive contexts (3) in the expert system. Then, it identifies the core state (7, 8) among the states inferring them (5, 6). After that, it checks the sensors and the conditions of the event (11, 12) that makes the transition (9, 10) to the core states. Finally, it checks whether there is the fact, the usedSameSensors (13), in the expert system.

Rule.2 Unnecessary Sensing Operation Search Rule ①

```
1:  ∃x1 ∃x2 ∃x3 ∃x4 ∃x5 ∃y1 ∃y2 ∃y3 ∃y4 ∃y5
2:  (context (id x1)) ∧ (context (id y1))
3:  ∧ (areExclusiveContexts (contextId x1) (contextId y1))
4:  ∧ (not (equal (x1) (y1)))
5:  ∧ (isInferredBy (contextId x1) (stateId x2))
6:  ∧ (isInferredBy (contextId y1) (stateId y2))
7:  ∧ (state (id x2) (isItCore "core"))
8:  ∧ (state (id y2) (isItCore "core"))
9:  ∧ (hasRangeState (eventId x3) (stateId x2))
10: ∧ (hasRangeState (eventId y3) (stateId y2))
11: ∧ (hasCondition (eventId x3) (sensorId x4) (conditionId x5))
12: ∧ (hasCondition (eventId y3) (sensorId y4) (conditionId y5))
13: ∧ (not (usedSameSensors (contextId x1) (eventId x3) (contextId y1) (eventId y3)))
    =>
14: (areUnnecessarySensingOperation (contextId x1) (eventId x3) (contextId y1) (eventId y3))
```

Figure 10. Rule 2: The unnecessary sensing operation search rule ①.

Second, Rule 3 finds the unnecessary sensing operation when some of sensors are of the same type in Figure 11. As mentioned in the previous section, in this case, the sensing conditions of the unnecessary sensing operation in exclusive contexts should not be disjointed (16).

Rule.3 Unnecessary Sensing Operation Search Rule ②

```
∃x1 ∃x2 ∃x3 ∃x4 ∃x5 ∃y1 ∃y2 ∃y3 ∃y4 ∃y5
(context (id x1)) ∧ (context (id y1))
∧ (areExclusiveContexts (contextId x1) (contextId y1))
∧ (not (equal (x1) (y1)))
∧ (isInferredBy (contextId x1) (stateId x2))
∧ (isInferredBy (contextId y1) (stateId y2))
∧ (state (id x2) (isItCore "core"))
∧ (state (id y2) (isItCore "core"))
∧ (hasRangeState (eventId x3) (stateId x2))
∧ (hasRangeState (eventId y3) (stateId y2))
∧ (hasCondition (eventId x3) (sensorId x4) (conditionId x5))
∧ (hasCondition (eventId y3) (sensorId y4) (conditionId y5))
∧ (equal (x4) (y4)
∧ (not (equal (x5) (y5)))
∧ (not (areDisjointedConditions(conditionId x5) (conditionId y5)))
  =>
(areUnnecessarySensingOperation (contextId x1) (eventId x3) (contextId y1) (eventId y3))
```

Figure 11. Rule 3: The unnecessary sensing operation search rule ②.

As an example of the problem definition in Section 3.1, the driving context and the seminar context and the driving context and the studying context are cases where the kinds of sensors are not the same. Therefore, all the sensing operations inferring these contexts at the same time are unnecessary sensing operations. However, in the case of the seminar context and the studying context, because all conditions of the sensors that have the same type are disjointed, there is no unnecessary sensing operation.

The expert system not only identifies the inefficient sensing operation but also defines rules to infer efficient sensing operation. In this paper, we propose a method of gradually delaying the period of sensing operation in exclusive contexts by using a linear backoff-algorithm that follows Rule 4 in Figure 12.

Rule.4 Efficient Sensor Operating Instruction Rule

```
∃x1 ∃x2 ∃x3 ∃x4 ∃y1 ∃y2
(or (areUnnecessarySensingOperation (contextId x1) (eventId x2) (contextId y1) (eventId y2))
(areUnnecessarySensingOperation (contextId y1) (eventId y2) (contextId x1) (eventId x2)))
∧ (or (sensor (id x1) (name "wifi"))
(sensor (id x1) (name "gps")))
∧ (hasCondition (eventId x1) (sensorId x3) (conditionId x4))
  =>
(linearlyBackoffSensingDuration (sensorId x3) (condition x4) (backoff 3) (max 9))
```

Figure 12. Rule 4: The efficient sensor operating instruction rule.

Since the rules for effective sensing methods could be modified and are extensible, the expert system continuously improves the efficiency of the sensing operation. In addition, the rules are shared and managed externally, so that more various methods can be proposed than those that are individually managed. Finally, the ExCore helps the application or middleware developer to perform efficient sensing operations considering the exclusive contexts.

5. Application and Evaluation

5.1. Application into Android

The ExCore's efficient sensor operating instruction should be modified and extended by the involvement of various developers. In order to satisfy such a requirement, we implemented the ExCore as a web service. Figure 13 is the architecture of the ExCore, and Figure 14 is the execution screen of the ExCore.

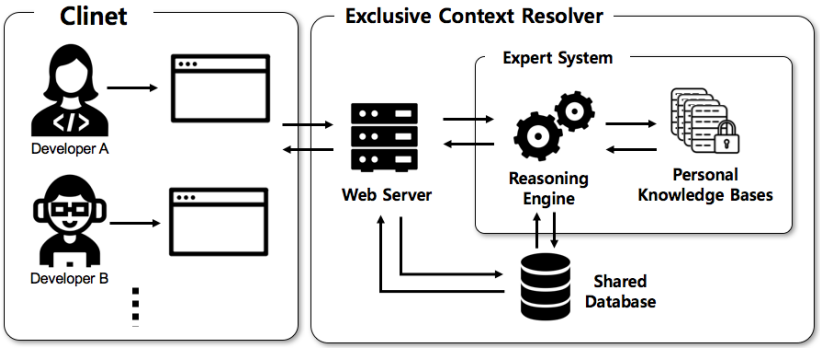

Figure 13. The architecture of the ExCore.

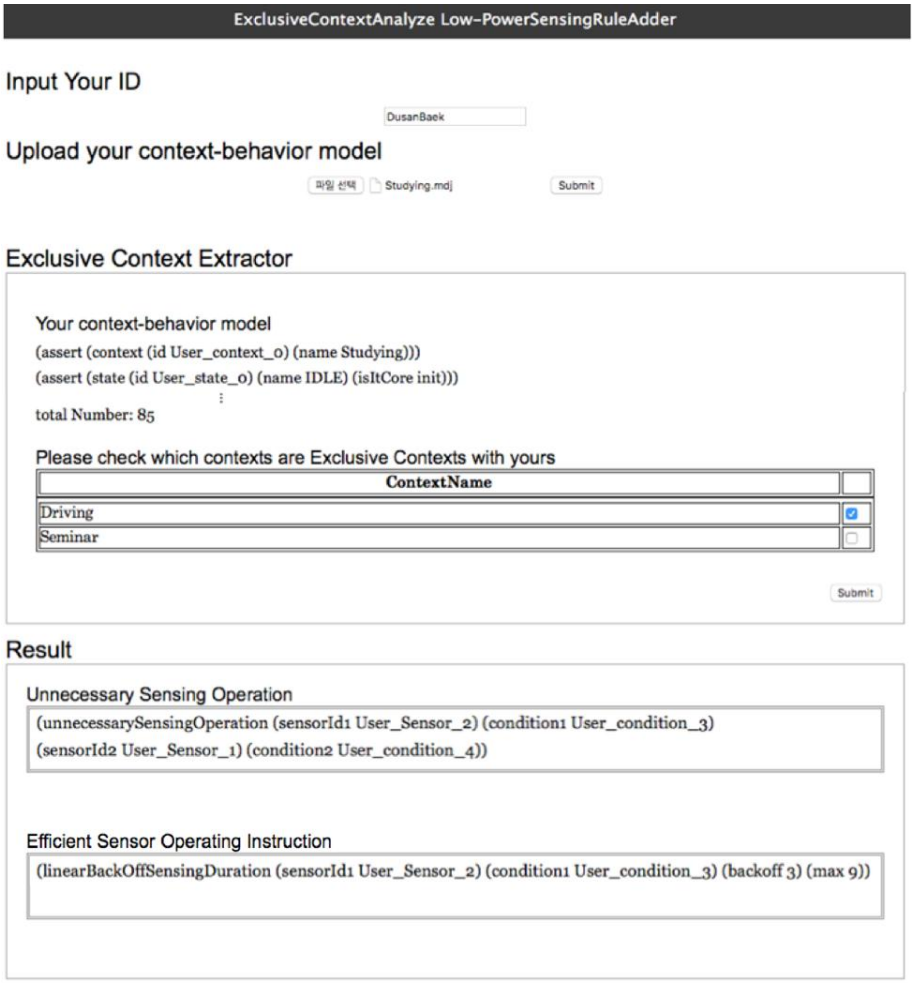

Figure 14. The execution screen of the ExCore.

Since the ExCore makes different inferences according to individual situations, each fact should be managed as a personal knowledge base. However, because the efficient sensor operating rule is generated through cooperation among various developers, they should be managed as a shred database. The developers are separated by id and submit their own context behavior model. After selecting the contexts that constitute the exclusive contexts with the input model, the ExCore identifies an unnecessary sensing operation and the developer receives efficient sensor operating instructions.

The expert system is designed to deal with a large number of facts compared with the number of facts used in this paper. Therefore, if the information about the sensors and their operation in the automata is accurate, the performance of running the automata against the sensor events would not be significantly affected.

5.2. Application and Middleware Using Efficient Sensor Operating Instruction

5.2.1. Environment

For evaluating the applicability and the feasibility of our system, we apply the efficient sensor operation instruction to the mobile application and middleware. The scenario for the evaluation is the same as the problem definition described in Section 3.1. We use the Nexus 4 for Android and Lollipop Version 5.1.0. For the measurement of power, we use the Portable Power Measurement and Analysis tool (PPAM) which was developed in previous work for measuring the power consumption of a mobile device. Because PPAM is a hardware-based measurement, not a model-based estimation, a more accurate measurement is possible [22]. We repeatedly measure the power consumption of each evaluation in our scenario while maintaining the same environment.

5.2.2. Application Using Efficient Sensor Operating Instruction

We have applied an efficient sensor operating instruction to the application. We have developed an application that performs unnecessary sensing operations in exclusive contexts. The application infers the contexts in our scenario using Wi-Fi, GPS, and the Volume sensor and consumes unnecessary power in the exclusive contexts. To solve the problem, we have modified the application by referring to the effective sensor operating instruction generated from the ExCore. The first proposed instruction is a linear backoff-algorithm that improves power efficiency by increasing the period of sensing when the unnecessary sensing operation is detected. The second proposed instruction is to use the 3-axis acceleration sensor to determine the next sensing time for avoiding the unnecessary sensing operation. Figure 15 shows the power consumption of the applications, and Figure 16 shows cumulative amount of difference of the power consumption between the default application and the modified applications using the efficient sensor operating instruction.

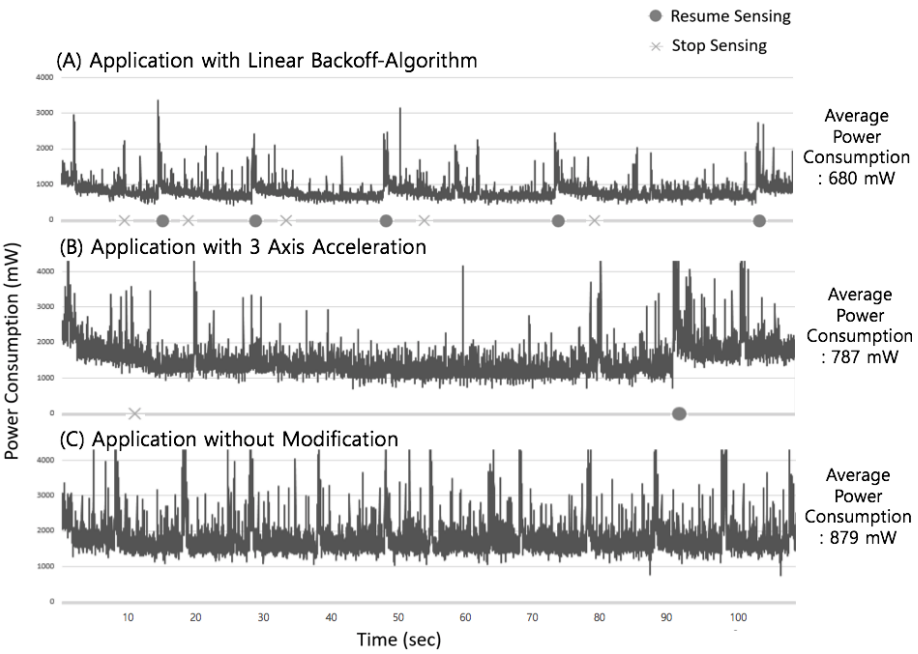

Figure 15. Power consumption of the context-aware application above the test scenario.

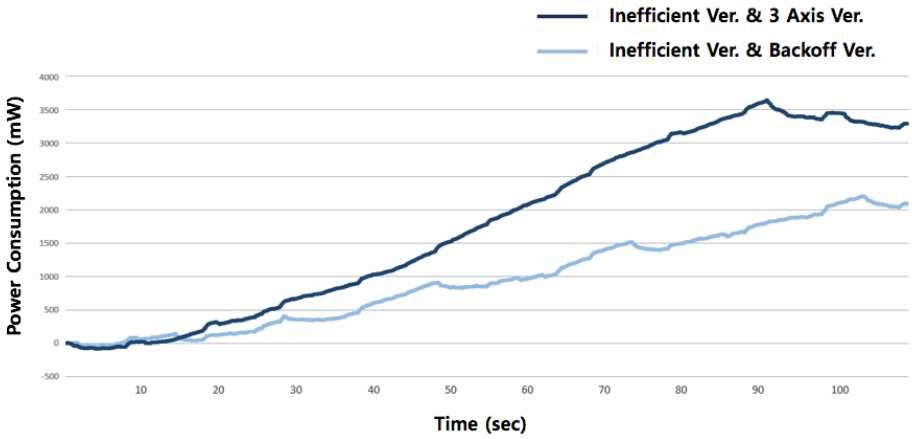

Figure 16. Cumulative amount of the difference of the power consumption between the tested applications.

As shown in Figure 15c, the default application cannot react in two or more exclusive contexts, and this leads to inefficient power consumption. Conversely, as shown in Figure 15a,b, the application can manage the sensor for efficient power consumption by increasing the period of sensing or by determining the next sensing time. In the case of Figure 15b, we adapt the linear backoff-algorithm to the application (10 s, 20 s, 30 s,..., max = 50 s). The result shows that the power efficiency is improved by 12% compared to the unmodified application, but the response time to the transition is postponed because of the increasing period of sensing. In the case of Figure 15a, we utilize the 3-axis sensor to determine the transition. The result shows that the power efficiency is improved by 29% compared to the unmodified application evaluated in the above scenario, and the response time is also as fast as the unmodified application. Figure 16 is a graph showing the cumulative amount of the power consumption in Figure 15. As shown in Figure 16, by adapting the efficient sensor operating instruction to the application, the application can save the power consumption continuously while staying in exclusive contexts.

5.2.3. Middleware Using Efficient Sensor Operating Instruction

It is very useful to adapt the efficient sensor operating instruction to the middleware of the OS because it allows for efficient sensing operation without modification of the application. In this evaluation, we have implemented a middleware that operates according to the instruction to cope with the exclusive contexts by using the 3-axis acceleration sensor introduced in Section 5.2.2. We have modified the Location Framework of the AOSP (Android Open Source Project) to control the operation of the Wi-Fi and the GPS sensor. The application used in this evaluation recognizes the exclusive context. If there is no movement for 1 min in the exclusive context, then it stops operation of the Wi-Fi and GPS sensor. On the other hand, the modified middleware operating through the instruction immediately stops operating the Wi-Fi and GPS sensor if movement is not detected by the 3-axis acceleration sensor in the exclusive contexts. Figure 17 is a graph comparing the power consumption between the default AOSP and the modified AOSP which reflects the efficient sensor operating instruction.

Although it is highly dependent on the scenario, modifying the middleware can improve the efficiency of the power consumption by about 62%. Furthermore, when modifying the middleware, it can apply efficient sensing operation in the exclusive contexts without modifying the individual application. However, modifying the middleware to control the sensor means that a modification of the OS is required. Therefore, it has limitations on being able to adapt to a non-commercial OS.

It can be seen that the power consumption is significantly lower when modifying the middleware compared to the application in Section 5.2.2. However, this is because the default power consumption of the commercial OS and AOSP is very different.

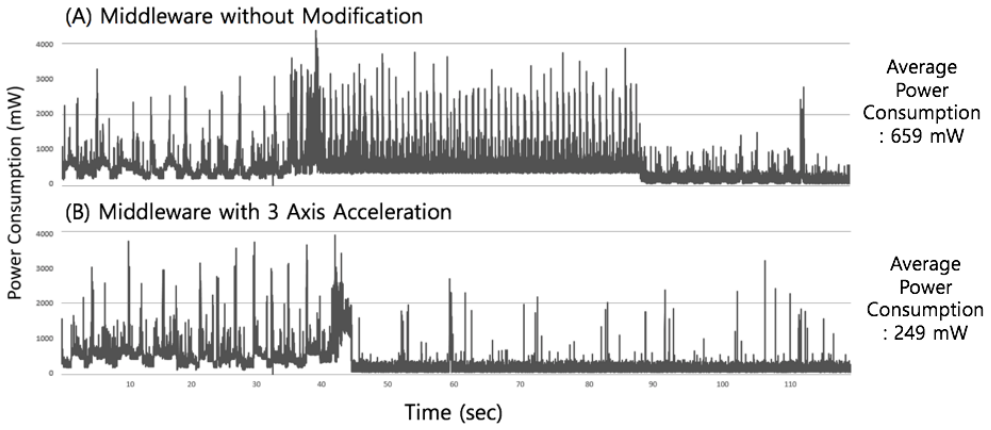

Figure 17. Comparison of the power consumption without and with modification of the middleware.

6. Conclusions

In this study, ExCore (Exclusive Contexts Resolver), which is a low-power sensing management system, was proposed to decrease inefficient power consumption in exclusive contexts. ExCore takes the sensor behavior model of the user and adds it as a fact to the expert system. This is followed by the identification of a sensing operation that infers the exclusive contexts through an efficiency search rule and presents a low-power sensing operation. The system was evaluated by examining the application and middleware with a low-power sensing operation that was proposed through the system. The results indicate an increase in the average power efficiency that corresponds to 12%–62% depending on the test scenario. However, if an application or middleware cannot control the sensor according to the instruction, power saving is impossible. Therefore, a future study will involve implementing a study to determine the exclusive contexts without user intervention and a middleware that is capable of accommodating a variety of low-power sensing operations.

Acknowledgments: This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2016-H8501-16-1006) supervised by the IITP (Institute for Information & communications Technology Promotion) and was also supported by the National Research Foundation of Korea (NRF) grant funded by the MSIP (NRF-2016R1A2B1014376).

Author Contributions: All authors contributed to the design of the proposed system and the writing of the paper. Jung-Won Lee contributed to the data modeling and the rule description. Dusan baek implemented the system and performed the experiment evaluation.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Albayram, Y.; Khan, M.M.H.; Bamis, A.; Kentros, S.; Nguyen, N.; Jiang, R. Designing challenge questions for location-based authentication systems: A real-life study. *Hum.-Centric Comput. Inf. Sci.* **2015**, *5*, 17. [CrossRef]
2. Vanus, J.; Smolon, M.; Martinek, R.; Koziorek, J.; Zidek, J.; Bilik, P. Testing of the voice communication in smart home care. *Hum.-Centric Comput. Inf. Sci.* **2015**, *5*, 15. [CrossRef]
3. Paek, J.; Kim, J.; Govindan, R. Energy-efficient rate-adaptive GPS-based positioning for smartphones. In Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, USA, 15–18 June 2010; pp. 299–314.
4. Saravanan, V.; Pralhaddas, K.D.; Kothari, D.P.; Woungang, I. An optimizing pipeline stall reduction algorithm for power and performance on multi-core CPUs. *Hum.-Centric Comput. Inf. Sci.* **2015**, *5*, 2. [CrossRef]
5. Pathak, A.; Hu, Y.C.; Zhang, M. Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices. In Proceedings of the 10th ACM Workshop on Hot Topics in Networks, Cambridge, MA, USA, 14–15 November 2011; p. 5.

6. Liu, Y.; Xu, C.; Cheung, S. Where has my battery gone? Finding sensor related energy black holes in smartphone applications. In Proceedings of the 2013 IEEE International Conference on Pervasive Computing and Communications (PerCom), San Diego, CA, USA, 18–22 March 2013; pp. 2–10.
7. Mei, H.; Lü, J. GreenDroid: Automated diagnosis of energy inefficiency for smartphone applications. In *Internetware*; Springer: Singapore, 2016; pp. 389–438.
8. Zhang, L.; Gordon, M.S.; Dick, R.P.; Mao, Z.M.; Dinda, P.; Yang, L. Adel: An automatic detector of energy leaks for smartphone applications. In Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Scottsdale, AZ, USA, 28 October 2010; pp. 363–372.
9. Wang, Y.; Lin, J.; Annavaram, M.; Jacobson, Q.A.; Hong, J.; Krishnamachari, B.; Sadeh, N. A framework of energy efficient mobile sensing for automatic user state recognition. In Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, Wroclaw, Poland, 22–25 June 2009; pp. 179–192.
10. Nath, S. ACE: Exploiting correlation for energy-efficient and continuous context sensing. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, Ambleside, UK, 25–29 June 2012; pp. 29–42.
11. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context aware computing for the internet of things: A survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454. [CrossRef]
12. Strang, T.; Linnhoff-Popien, C. A context modeling survey. In Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management at UbiComp 2004, Nottingham, UK, 7 September 2004.
13. Bellavista, P.; Corradi, A.; Fanelli, M.; Foschini, L. A survey of context data distribution for mobile ubiquitous systems. *ACM Comput. Surv. (CSUR)* **2012**, *44*, 24. [CrossRef]
14. ARM big. LITTLE Technology. Available online: https://www.arm.com/products/processors/technologies/biglittleprocessing.php (accessed on 26 January 2017).
15. Shen, H.; Balasubramanian, A.; LaMarca, A.; Wetherall, D. Enhancing mobile apps to use sensor hubs without programmer effort. In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, Osaka, Japan, 7–11 September 2015; pp. 227–238.
16. Android Developers. Available online: http://developer.android.com/reference/ (accessed on 26 January 2017).
17. Pathak, A.; Hu, Y.C.; Zhang, M. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In Proceedings of the 7th ACM European Conference on Computer Systems, Bern, Switzerland, 10–13 April 2012; pp. 29–42.
18. Hao, S.; Li, D.; Halfond, W.G.; Govindan, R. Estimating mobile application energy consumption using program analysis. In Proceedings of the 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 92–101.
19. Lee, S.; Jung, W.; Chon, Y.; Cha, H. EnTrack: A system facility for analyzing energy consumption of Android system services. In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, Osaka, Japan, 7–11 September 2015; pp. 191–202.
20. Nikzad, N.; Chipara, O.; Griswold, W.G. APE: An annotation language and middleware for energy-efficient mobile application development. In Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 515–526.
21. Baek, D.; Park, J.; Lee, B.; Lee, J. A Low-Power Sensing Management Method for Sustainable Context-Awareness in Exclusive Contexts. In Proceedings of the International Conference on Computer Science and its Applications, Bangkok, Thailand, 19–21 December 2015; Springer: Singapore, 2016; pp. 950–956.
22. Choi, K.-Y.; Lee, J.-W. Portable power measurement system for mobile devices. *J. KIISE Comput. Pract. Lett.* **2014**, *3*, 131–142.