International Journal of Digital Content Technology and its Applications
Vol. 2 No 1, March 2008

Specifying a WS-ECA Working Framework for Ubiquitous Web Services in Object-Process Methodology

Haining Lee[1], Jaeil Park[1]*Corresponding author, Peom Park[1], Dongmin Shin[2]
[1] *Industrial & Information System Engineering, Ajou University, San 5, Woncheon-dong, Yeongtong-gu, Suwon, 443-749, Korea*
[2] *Department of Information and Industrial Engineering, Hanyang University, Ansan. 1217, Kyunggi-do, Korea*
leehaining@ajou.ac.kr, jipark@ajou.ac.kr, ppark@ajou.ac.kr, dmshin@hanyang.ac.kr

Abstract

The Web Services-Event-Condition-Action (WS-ECA) framework is designed to enable the heterogeneous communication devices in Ubiquitous Computing Environments to achieve inter-operability via event-driven coordination. Object-Process Methodology (OPM) is an integrated modeling method that unifies the function, structure and behavior of a system. This paper presents specifying a WS-ECA framework for ubiquitous web services in OPM. In addition, we propose a dynamic conflict detection and resolution method in which dynamic conflicts are detected by examining the post-conditions of services being triggered and are resolved by acquiring user priority, service priority, and user's service preference.

1. Introduction

Ubiquitous computing is a post-desktop model of human-computer interaction in which information processing has been enhanced by making computing access everywhere throughout a dynamic set of small networked devices [1]. Devices with particular functions and services in ubiquitous computing environments provide their own services by interacting with each other via various types of network.

Web service technologies (WST) are rapidly emerging as an effective means for achieving inter-operability in ubiquitous computing environments thanks to their well-defined and widely accepted Internet protocol [2]. It has become a de facto standard for integrating business applications [3]. The use of web service technologies in ubiquitous computing environments that requires timely notification of services available raises the need of developing the most effective coordination method among services.

Event-Condition-Action (ECA) rules have been introduced in database systems to facilitate event-driven coordination. Ubiquitous web service coordination has taken the ECA rule-based approach to coordinate various distributed devices in ubiquitous computing environments [4]. To define various web services and their interactions throughout distributed devices, in particular, the Web Services-ECA (WS-ECA) rule description language is introduced, which can abstract a wide variety of reactive behaviors of various ubiquitous web services as well as service interactions [5]. Therefore, distributed devices embedded with ECA rules described in WS-ECA language can be triggered by event and provide their services [6].

The WS-ECA rules registered for ubiquitous web services can be requested simultaneously by multiple users. At build-time and run-time, some rules may have inconsistencies with others, which cause conflicts when triggered and executed. The notion of a service constraint set, which is a set of not being allowed to occur at the same time, is proposed for the dynamic conflict detection and resolution method [7]

The OPM is a useful methodology to graphically specify systems in a single unified model that describes the static-structural and behavioral-procedural aspects of systems [8]. The building blocks of OPM are objects, processes, states, and structural and behavioral links and are represented in a set of graphical diagram and text [9-10]. In this paper, we use OPM to specify a WS-ECA framework for ubiquitous web services. In addition, we propose a conflict detection method by examining the post-conditions of services being triggered and a resolution method by acquiring user information regarding user priority, service priority, and user's service preference. In Section 2, the WS-ECA framework for ubiquitous web services is presented, and Section 3 briefly introduces OPM. Section 4 shows how the WS-ECA framework is specified in OPM and proposes a conflict detection and resolution method for ubiquitous service management. Section 5 concludes the paper.

2. WS-ECA framework

ECA rules perform actions in response to events when stated conditions are satisfied. Likewise, WS-ECA rules consist of events, conditions and actions: 1) events are notification messages from services or users, 2) conditions are a Boolean expression that must be satisfied in order to activate devices, and 3) actions are instructions that invoke services or generate events [7].

Service devices distributed in ubiquitous computing environments provide specific web services or generate events via publish/subscribe mechanism of WS-Eventing and the web service invocation based on SOAP messaging [11-12]. Those service devices can be specified and managed using an XML-based WS-ECA rule description. That is, WS-ECA rules enable service devices to activate sequentially or concurrently via event based interaction.

To satisfy requirements that are necessary for effective coordination among ubiquitous service devices, WS-ECA rules are designed such that the WS-Eventing module collects and delivers event notification to devices. Devices interact with each other over event notifications. When an ECA rule is triggered by an event and its condition is satisfied, the corresponding device sends a service request to the service managing module which manages conflict detecting and resolving, service performing and service monitoring. If no conflict is detected, the service request will be approved.

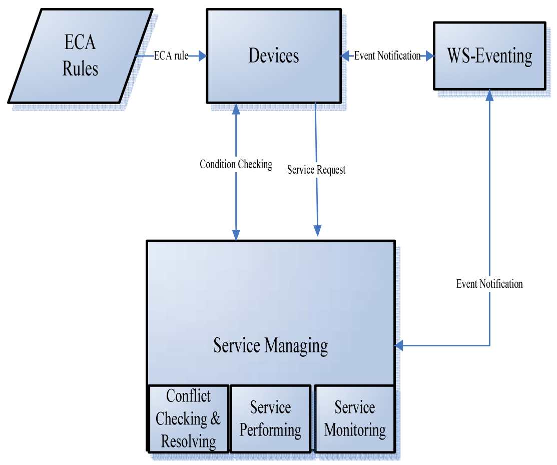

Figure 1. WS-ECA framework

3. The Object-Process Methodology

The Object-Process Methodology (OPM) incorporates the static-structure and behavioral procedural aspects of a system into a single unifying model. The basic premise of OPM is that objects and processes are two types of equally important classes of things, which together describe the function, structure and behavior of a system in virtually any domain. The OPM specifies an object class with states. At any point in time, each object is at one state and is moved to other state through the occurrence of a process.

OPM unifies the system lifecycle stages-specification, design and implementation-within one frame of reference, using a single diagramming tool- a set of Object-Process Diagrams (OPDs) and a corresponding subset of constrained natural language, called Object-Process Language (OPL). A set of OPDs or its corresponding OPL script completely specifies a system [9]. Table 1 shows the main OPM elements, symbols and semantics [9] and Figure 2 illustrates how OPDs are used to specify the wedding example and shows resultant OPL.

Table 1. Main OPM elements, their symbols and semantics

Element Name	Symbol	Semantics
Object	▭	A thing that has the potential of unconditional existence
Process	⬭	A pattern of transformation that objects undergo
Environmental thing	▭ ⬭	An environmental (external) thing (object or process) which communicates with the system
Characterization	◬	A relation representing that a thing (object or process) exhibits another thing
Generalization	△	A relation denoting the fact that a thing generalizes a set of specialized things
Aggregation	▲	A relation which denotes that a thing consists of other things
General structural relationship	⇌	A general association between things
Instrument link	o—	A link indicating that a process requires an (input) object for its execution
Effect link	◁—▷	A link indicating that a process changes an object
Result/ Consump-tion link	◁—	A link indicating that a process creates/consumes an object
Invocation link	◁↯	A link indicating that a process activates (invokes) another process
Condition link	ⓒ—	A link representing a condition required for a process execution. While an enabling link has a "wait until" meaning, a condition link has an "if" meaning
Agent link	●—	A link indicating that an external agent is required for the process execution

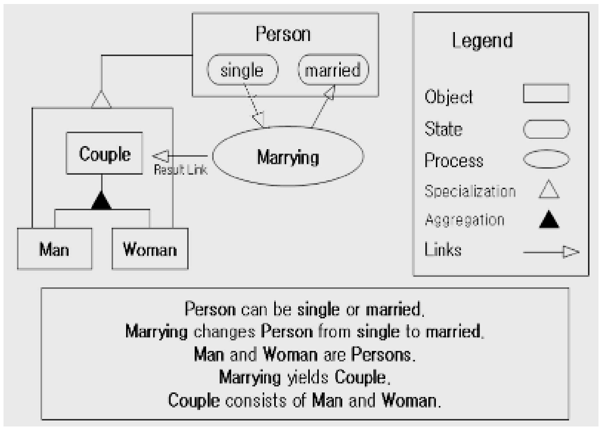
Person
single
married
Legend
Object
State
Process
Specialization
Aggregation
Links
Couple
Marrying
Result Link
Man
Woman
Person can be single or married.
Marrying changes Person from single to married.
Man and Woman are Persons.
Marrying yields Couple.
Couple consists of Man and Woman.

Figure 2. OPDs and OPL script of wedding example in OPM

4. Specifying a WS-ECA working framework for ubiquitous web services in OPM

The WS-ECA working framework for ubiquitous web services can be efficiently specified by OPM. Figure 3 shows a top-level OPD specifying the WS-ECA working framework, which presents the triggering and conflict detecting processes of ECA rules. It shows all the objects and processes which take part in this framework: Global Rule Manager, ECA Rules, and Activated Services. Global Rule Manager keeps conflict and resolution rules and enables Build-Time Checking, Run-Time Conflict Detecting, and Conflict Resolving.

Build-time Conflict Checking and Run-time Conflict Detecting processes detect conflicts between rules at build-time and run-time respectively according to the conflict rules that are kept in Global Rule Manager. Build-time Conflict Checking examines if a set of rules violates conflict rules when registering ECA Rules. Triggering Process is enabled by an event generated by User and at the same moment invokes Run-Time Conflict Detecting. If the outcome of Run-Time Conflict Detecting is true, Conflict Resolving Process is enabled. Otherwise, Triggered Process is activated, which would change the state of Devices and/or Environment.

In WS-ECA framework, all devices are described and coordinated by using ECA rules. These rules are stored and managed by ECA Rules. Activated Services is a set of running services and used for detecting run-time conflicts. In what follows, the WS-ECA framework is further in-zoomed to show sub-processes including Run Time Conflict Detecting and Conflict Resolving.

4.1 ECA Rules

ECA rules are used to express control flows for ubiquitous web services. These rules specify each action (Triggering Process), its triggering event (Event), and its guarding condition (Activating Condition). An action is executed when a triggering event occurs, if and only if the guarding condition is fulfilled. Figure 4 is the OPD of an ECA rule. All devices providing services are registered in ECA Rules.

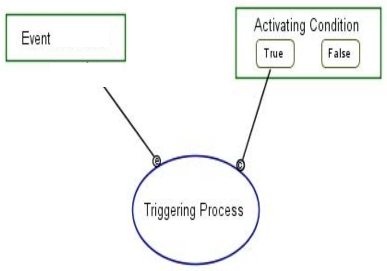

Figure 4. OPD of an ECA rule

An ECA Rule is composed of Event, Condition and Action. Figure 5 is a blow-up of ECA Rules.

- Event: an event is the incident which triggers a rule. It has three types: 1) External events, which are generated by devices; 2) Internal events, which are generated by the internal system components, and 3) Time events, which are generated by a timer at some specific point or period of time.
- Condition: the condition part of an ECA rule is a Boolean statement that must be satisfied to activate a rule. The condition statement is expressed in terms of an XPath expression in which Event Variables, Environment Variables and Device Variables are used. The environment variables and device variables are provided by devices and environment.
- Action: the action part of a ECA rule is the actions that must be executed when a triggered rule is activated. Actions can be one or combination of the following types: 1) Service Invoking, which invokes a service, 2) ExEvent Creating, which creates external event, 3) IntEvent Creating, which creates internal event.

Devices and Environment provide their variables to ECA Rules such as state variables that are used to detect conflicts by Global Rule Manager.

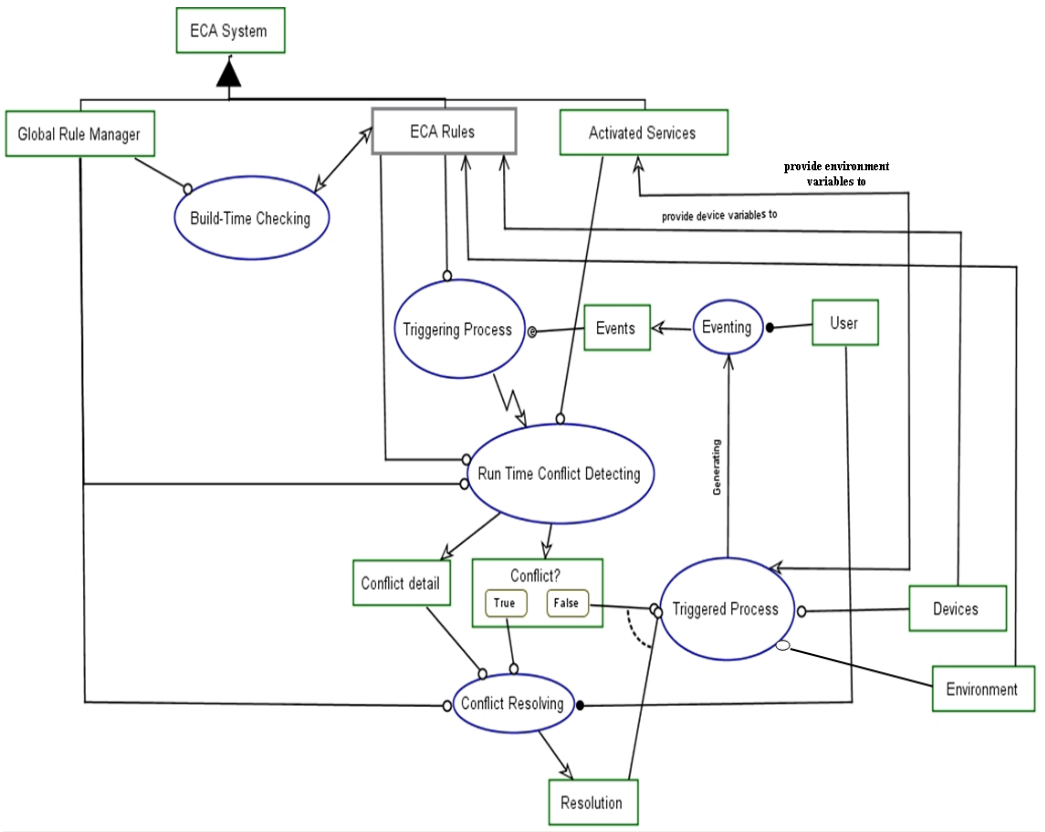

Figure 3. Top Level OPD of the WS-ECA framework

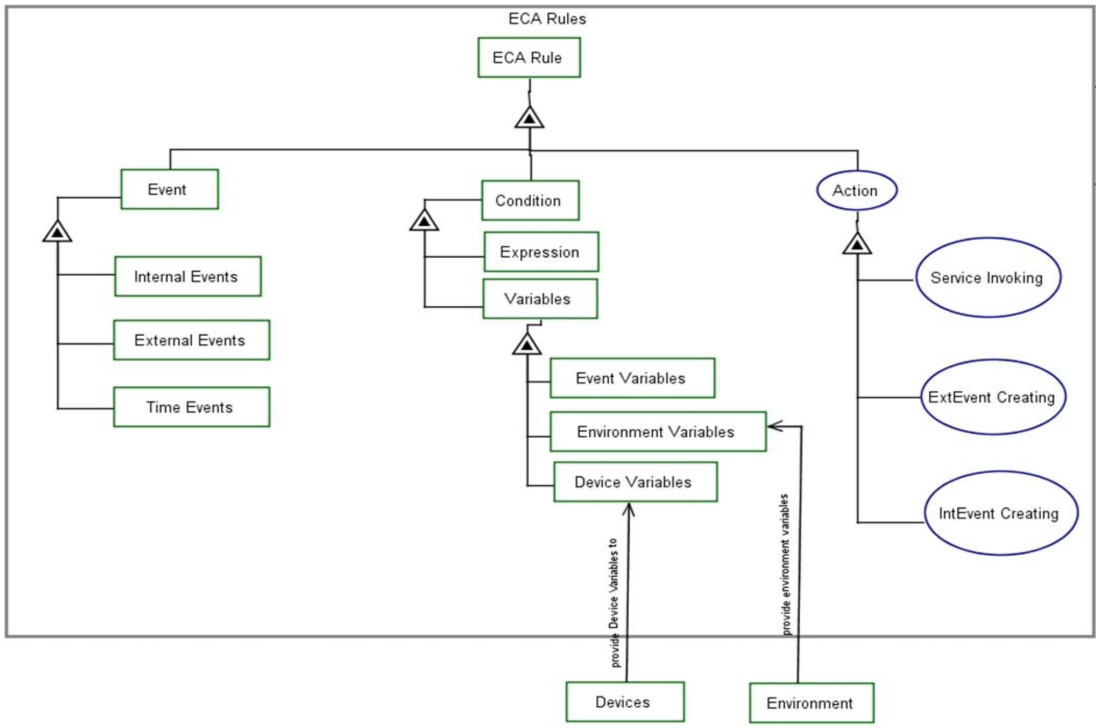

Figure 5. OPD of ECA Rules

4.2 Global Rule Manager

Global Rule Manager has Conflict Rules and Resolution Rules. Conflict Rules define potential conflicts between different services and are used to detect ECA rules' conflicts at build-time (static conflicts). Dynamic conflicts are detected by examining the post-conditions of existing and new requested services. A service of opening a window while an air conditioner is currently working is a representative example of static conflicts. The following is the corresponding xml:

```
<Conflict Rules>
   <Conflict Rule>
        <conflict name='air conditioner conflict'>
           <A>air conditioner=on</A>
           <B>window=open</B>
        </conflict>
   </Conflict Rule>
</Conflict Rules>
```

Resolution Rules provides a means for expressing various ways to resolve dynamic conflicts. User priority, service priority, and user's service preference comprise Resolution Rules. The detail of the use of Resolution Rules in Global Rule Manager is discussed in Section 4.5.

4.3 Activated Services

Activated Services is a set of running services. When a service is activated, the service is registered to Activated Services. These are used to detect run-time conflicts by Run-Time Conflict Detecting. For example, when Rock Music Playing Service whose service id is 001 is activated at 17:00 and is expected to last for 30minutes (1800seconds), the service registers its service information to Activated Services. The following is the corresponding xml:

```
<Activated Service Info>
   <Service name=Rock Music Playing id=001
time=17:00 period=1800 />
</Activated Service Info>
```

4.4 Run-Time Conflict Detecting

With post-conditions, the effects of Triggering Process can be predicted. Device and Environment register their states to ECA rules. The states can be categorized into pre- and post-state. When an event occurs in a situation when condition is true, and there is no conflict found, then the triggered action would change the pre-state of Device and/or Environment to its post-state. These post-states affected by Triggered Process are named post-condition. Figure 6 shows the post-conditions of triggered process.

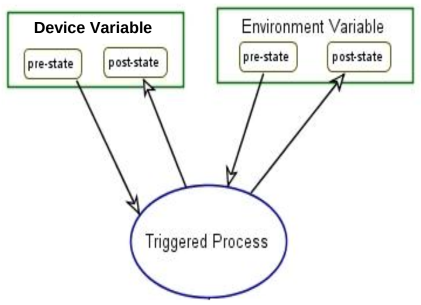

Figure 6. Post conditions of triggered process

As shown in Figure 3, Triggering Process invokes Run-Time Conflict Detecting, which is a process of getting the post-conditions of triggering process from ECA Rules and comparing them with the post-conditions of services running in Activated Services. If a conflict is found, Conflict Notification is generated. If no conflicts are found, Triggering Process initiates Triggered Process that changes the state of Device and Environment Variable. Figure 7 is a blow-up of Run-Time Conflict Detecting of the top level OPD of the WS-ECA framework. For example, when Jack enters room (Event "Jack enters room" occurs), if there is no music playing (Condition "No music is playing" is true), process "Rock Music Playing" would be triggered (Triggering Process). The rock music service is modeled in OPD as shown in Figure 8.

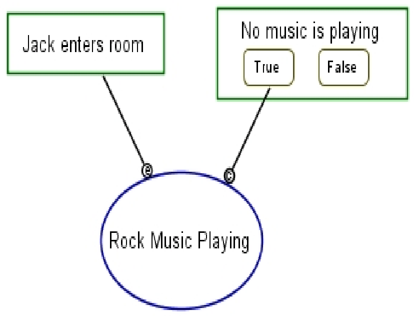

Figure 8. Rock music playing service example

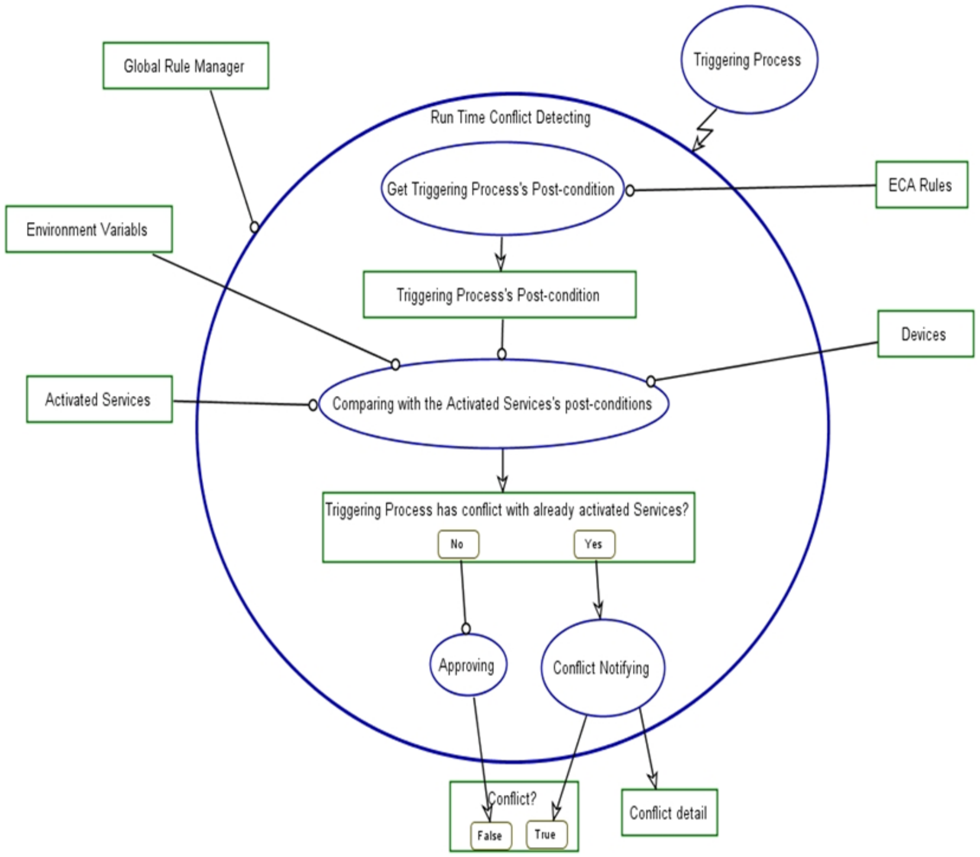
Global Rule Manager
Triggering Process
Run Time Conflict Detecting
Get Triggering Process's Post-condition
ECA Rules
Environment Variabls
Triggering Process's Post-condition
Devices
Activated Services
Comparing with the Activated Services's post-conditions
Triggering Process has conflict with already activated Services?
No
Yes
Approving
Conflict Notifying
Conflict?
False
True
Conflict detail

Figure 7. Run-time conflict detecting process

If the rock music playing service is activated, the power state of a CD player will be set to "on", the sound intensity will be changed to 40db, and the type of music will be set as "rock music". Before activating, this rock music playing service invokes Run-Time Conflict Detecting to check if there is a potential conflict at run-time. The post-conditions of rock music playing (Triggered Process) and their xml are shown in Figure 9.

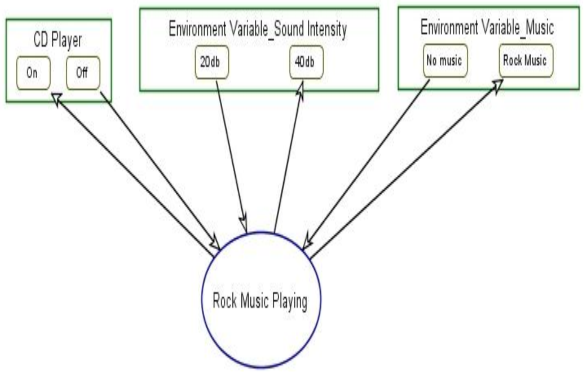
CD Player
On
Off
Environment Variable_Sound Intensity
20db
40db
Environment Variable_Music
No music
Rock Music
Rock Music Playing

Figure 9. Post-conditions and their xml of rock music playing service

```
<Post conditions>
     <Post condition service_name=Rock Music Playing
            service_id=001>
        <aim user name=Jack id=101 />
         <device>
            <device_sate name=CD player's power
                        id=010 value=on />
         </device>
         <environment>
              <environment_variable
            name=sound_intensity id=020 value=40db/>
              < environment_variable name=music_type
                        id=021 value=rock>
         </environment>
     </Post condition>
</Post conditions>
```

If a keep quiet service is already running with the following post-condition:

<environment_variable name=sound_intensity id=020 value=<20db/>

Obviously, the post-condition of the rock music playing, "sound intensity=40db" is conflicting with that of the keep quite service, "sound intensity<20db". The

Service-Service conflict is detected. This conflict can be resolved, which is discussed next.

4.5 Conflict Resolving

Conflicts occur because of different services competing on the same exclusive resources or environmental variables. Avoiding potential conflicts is the best strategy to resolve them. If potential conflicts can not be defined and avoided, however, it relies on the conflict resolution rule.

4.5.1 Conflict Avoiding

Our strategy for conflict avoiding is using a "no disturb" slip to show the exclusive use of a device or environmental variable (temperature, luminance, etc). Whenever a service is using a device or an environmental variable and does not want to be disturbed, it should register a "no disturb" slip to Global Rule Manager. When other services try to use the device or environmental variable with "no disturb" slip, Global Rule Manager would inform that they are already in exclusive use and can not be disturbed. This slip avoids potential conflicts. If no "no disturb" slip is registered for a service, the service can be conflicting with other requesting services. To address this conflict, a conflict resolving method is needed.

4.5.2 Conflict Resolution Rule

The conflicts between different services can be classified into two types: 1) Single User Conflict: When conflicting services are provided to the same user, this type of conflict is called Single User's Service-Service Conflict and 2) Multi-Users Conflict: When conflicting services are provided to different users, this conflict is called Multi-Users Conflict. These conflicts can be resolved by acquiring user priority, service priority, and user's service preference.

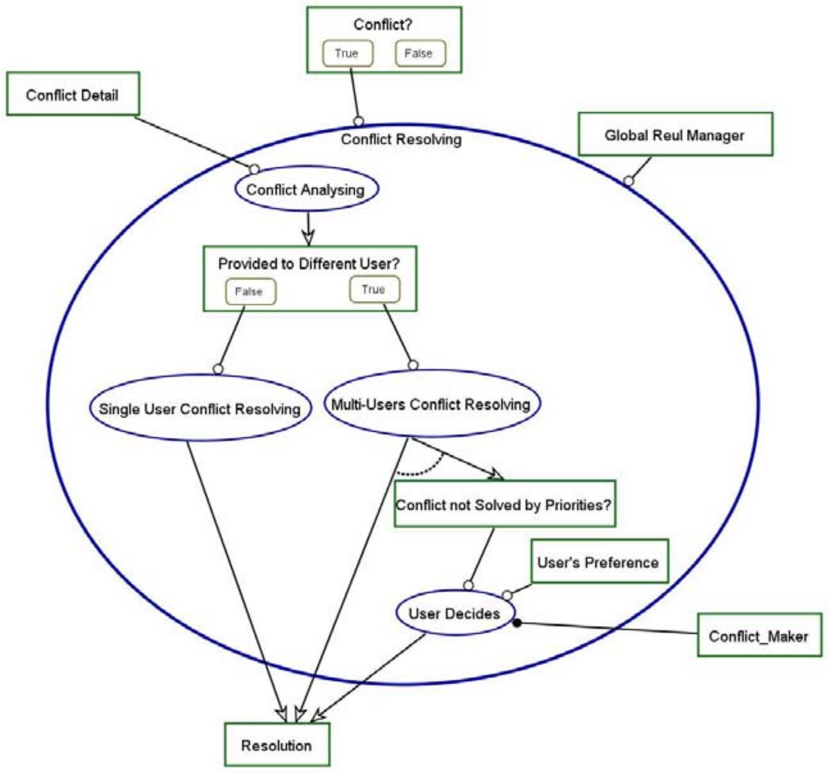

Figure 10. Conflict Resolving Process

Single User Conflict Resolving

Single User Conflict can be resolved by executing the service with higher priority. If conflicting services have the same priority, Global Rule Manager would check user's service preference record. The service which is compatible with user's preference would be executed. When there is no user's preference record, Global Rule Manager would let the user decide which service to be executed.

Multi-Users Conflict Resolving

When two services providing to different users are conflicting, the service requested by higher user priority would be executed. If both users have the same priority, the service with higher service priority is executed. Once conflict can not be solved by user priority and service priority, the decision will be made by the conflict-maker who invokes the conflicting service. When there's no conflict-maker's service preference record, Global Rule Manager resolves the conflict according to conflict-maker's service preference or decision.

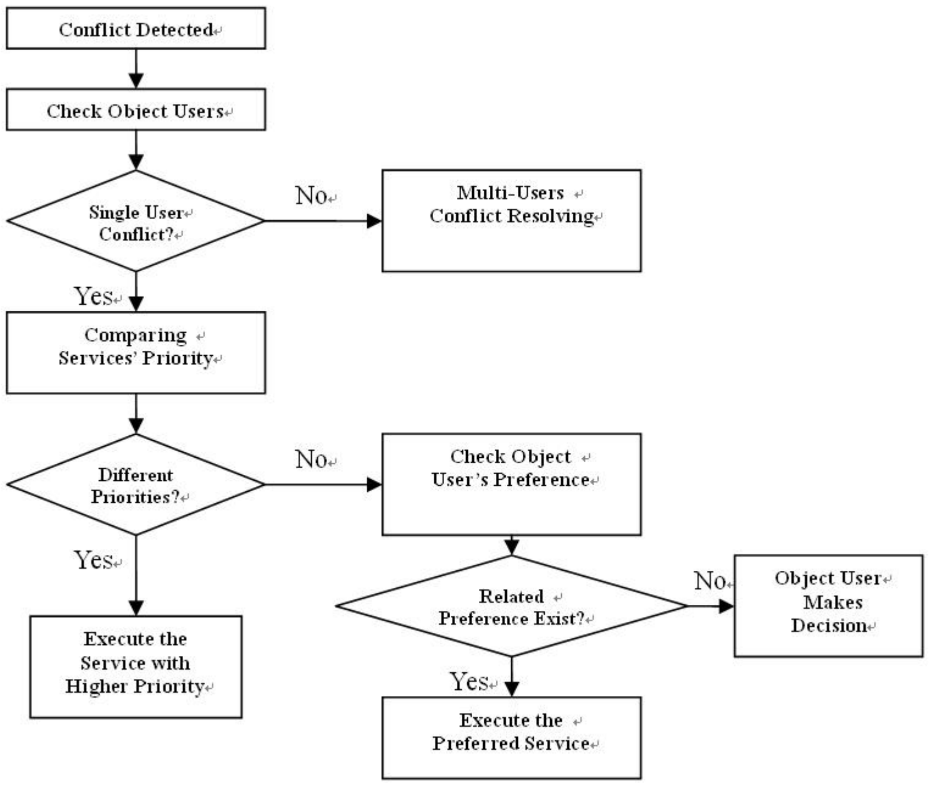
Conflict Detected
Check Object Users
Single User Conflict?
No
Multi-Users Conflict Resolving
Yes
Comparing Services' Priority
Different Priorities?
No
Check Object User's Preference
Yes
Execute the Service with Higher Priority
Related Preference Exist?
No
Object User Makes Decision
Yes
Execute the Preferred Service

Figure 11. Single User Conflict Resolving

User priority, service's priority, user's service preference are kept in Conflict Resolution Rule. An example of xml description of Conflict Resolution Rule is shown below:

```
<Conflict Resolution Rule>
    <User priority>
        <user name=Dad id=101 priority=001 />
        <user name=Mom id=102 priority=001 />
        <user name=Child id=103 priority=002 />
    </User priority>
    <User's preference>
        <user neme=Dad id=101>
            <environment_variable name=light id=020
value=on/>
        </user>
    </User's preference>
    <Service priority>
        <service name=sleep id=011 priority=001>
        <service name=lighting id=012 priority=001>
        <service name=music id=013 priority=002>
    </Service priority>
</Conflict Resolution Rule>
```

This resolution rule describes that Dad and Mom have the same user priority (001) which is higher than Child's priority (002). Dad prefers the light to be on. And Sleep Service's Service Priority is the same with Lighting Service, which is higher than the Music Service. When Mom's sleeping, Sleep Service is activated. A conflict occurs when Child wants to listen to the music and activates Music Service. Global Rule Manager checks Conflict Resolution Rule that Mom's priority is higher than Child's. Then Child's Music Service will be prevented unless Sleep Service stops.

When Dad comes in at night and tries to turn on the light, if Mom's Sleep Service already has a "No Disturb" slip, the light would not be turned on and potential conflict is avoided. If there is no "No Disturb" slip registered, Global Rule Manager checks Dad's User Preference and finds that Dad's priority can not solve the conflict since Dad and Mom have the same priority. According to Dad' service preference, Lighting Service would be activated because Dad prefers light to be on while Sleeping Service on.

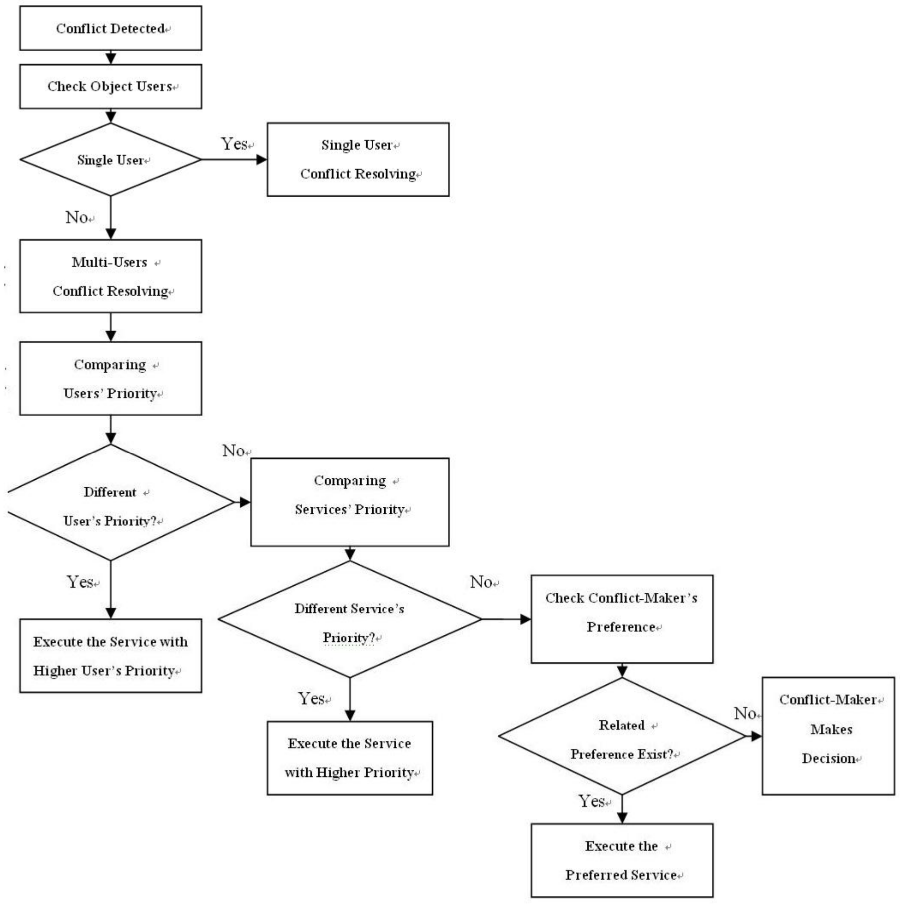
Conflict Detected
Check Object Users
Single User
Yes
Single User Conflict Resolving
No
Multi-Users Conflict Resolving
Comparing Users' Priority
Different User's Priority?
No
Comparing Services' Priority
Yes
Execute the Service with Higher User's Priority
Different Service's Priority?
No
Check Conflict-Maker's Preference
Yes
Execute the Service with Higher Priority
Related Preference Exist?
No
Conflict-Maker Makes Decision
Yes
Execute the Preferred Service

Figure 12. Multi-Users Conflict Resolving

5. Conclusion

In this paper we present specifying a WS-ECA framework for ubiquitous web services in Object-Process Methodology (OPM), which presents that the WS-ECA framework can be easily specified and understood as a whole in OPM via a set of Object Process Diagrams. In addition, we propose a practical conflict detection method based on the post-conditions of services being triggered and user information. By comparing the post-conditions of various services with conflict rules, potential conflicts can be detected. To resolve conflicts, we rely on user information regarding users' priority, service priority, and user's preference. Also "no disturb" sign strategy is used to avoid potential conflicts.

Our future work will address the resolution rules for conflict resolution and focus on its application to the health care domain. In addition, we will make an attempt to developing auto-code generation from OPDs to a practical programming language in order to generate WS-ECA rules in xml format.

6. Acknowledgements

This research is supported by the Ubiquitous Computing and Network (UCN) Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea

7. References

[1] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", IEEE Personal Communications, 2001, pp. 10 - 17.

[2] A. Sashima, N. Izumi, and K. Kurumatani, "Location-Mediated Coordination of Web Services in Ubiquitous Computing", Proceeding of IEEE Int'l Conf. WebServices (ICWS'04), 2004, pp. 109-114

[3] S. Vinoski, " Integration with Web Services", IEEE internet computing, 2003, vol. 7(6), pp.75-77

[4] J. Bailey, A. Poulovassilis, and P. Wood, "An Event-Condition-Action Language for XML", Proceedings of the 11th Int. Conf. on the World Wide Web, 2002.

[5] W. Lee, S. Lee, and K. Lee, "Conflict Detection and Resolution Method in WS-ECA Framework", ICACT 2007

[6] J. Jung, S. Han, J. Park, and K. Lee, "WS-ECA: An ECA Rule Description Language for Ubiquitous Services Computing", Edinburgh, UK, 2006

[7] J. Jung, J. Park, S. Han, and K. Lee, " An ECA-based Framework for Fecentralized Coordination", Information and Software Technology (2006), in press

[8] D. Dori, Object-Process Methodology: a holistic System paradigm, Springer; New York, 2002

[9] D. Dori, "Object-process Analysis: Maintaining the Balance Between System Structure and Behaviour", Journal of Logic and Computation 1995 5(2):227-249.

[10] A. Sturm, D. Dori, and O. Shehory, "Single-Model Method for Specifying Multi-Agent. Systems", AAMAS 2003.

[11] J. Hanson, "Event-driven services, in: SOA: design an event-driven and service-oriented platform with Mule", JavaWorld, 2005. Available from:
<http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html/>.

[12] B. Michelson, "Event-driven architecture overview: event-driven SOA is just part of the EDA story", White paper, Patricia Seybold Group, 2006. Available from:
http://www.psgroup.com/detail.aspx?ID=681/

Authors' bio.

Jaeil Park is an Assistant Professor of Industrial and Information Systems Engineering at Ajou University. He received his Ph.D. in 2005 in Industrial and Manufacturing Engineering from the Pennsylvania State University, his M.S. in 1997 in Iron and Steel Technology and his B.S. in 1995 in Mechanical Engineering from Pohang Science and Technology.

Dongmin Shin received the B.S. and the M.S. degrees in industrial engineering from Hanyang University, Seoul, Korea, in 1994 and Pohang University of Science and Technology (POSTECH) in 1996, respectively. He earned his Ph.D. degree in The Department of Industrial and Manufacturing Engineering at the Pennsylvania State University in 2005.

Peom Park received his Ph.D. degree in IMSE of Iowa State University and is a Profess of Industrial & Information Systems at Ajou University since 1995. He is currently a CEO of Humintec CO., Ltd working on Ubiquitous Telemedicine HCI (UCN-Korea) and Mobile & RFID POC(MIC-Korea).

Haining Lee received his B.S. degree in Electronics from Tsinghua University in China and is currently a master student in Industrial & Information Systems at Ajou University.