Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management
Kuala Lumpur, Malaysia, January 22 – 24, 2011

A Ubiquitous Process Coordination System for RFID/USN Events

Kyuri Kim, Kyuhyup Oh, Pablo Rosales, Jae-Yoon Jung
Department of Industrial and Management Systems Engineering
Kyung Hee University, Yongin-si, Gyeonggi-do, 446-701, Republic of Korea

Abstract

Today, many companies develop business integration environments with a variety of ubiquitous computing devices. Nevertheless, existing business process studies still have difficulty about the technology which can manage and control by integration environment of the occurrence events in real-time. In this paper, we introduce a process coordination system to support the processing and analysis of ubiquitous events such as RFID and USN. To implement the ubiquitous process coordination system, we adopt Complex Event Processing (CEP) technology and Event-Condition-Action (ECA) rules. The CEP engine monitors real-time event occurrences and the rule engine triggers proper rules according to the captured events.

Keywords
RFID, USN, event, business process, CEP, ECA rules

1. Introduction
A variety of devices and ubiquitous computing systems such as RFID (Radio Frequency Identification), USN (Ubiquitous Sensor Network) and mobile devices generate and exchange significant events during their execution in ubiquitous environment. Many companies currently develop business integration environments based on service-oriented architectures; however, such approach is not enough for the seamless business process integration with environment-sensing equipment in distributed and heterogeneous environments.
In this paper, we introduce a ubiquitous process coordination system for RFID and USN event processing. The goal of the system is to illustrate how to integrate real-time events to process coordination systems by adopting Complex Event Processing and event-driven rule technologies. For this purpose, we developed a process coordination system, named edUFlow, for integrating USN and RFID to the rule-based process engine. We first developed ubiquitous environment for processing real-time complex event by using web services and XML. Next, we designed a process coordinator system for integration management of the distributed events on ubiquitous environment. Finally, we implemented a rule-based system which can monitor complex real-time events on integration environment using the ECA-based process.
This paper is organized as follows. In Section 2 we discuss related work on event-driven BPM and service coordination. The proposed ubiquitous process coordination is presented in Section 3. The edUFlow system implementation is then described in section 4. Finally, Section 5 concludes the paper.

2. Related Work
Business process management plays a critical role in bridging the divide between business and information technology. Service-oriented architecture has recently been supplemented with event-driven architecture in order to reflect a variety of real-time events which are generated from distributed and heterogeneous information systems. Ammon et al. investigated business values of event-driven business process management [1]. Henglein et al. presented the POET (process-oriented event-driven transaction) system to illustrate enterprise system architecture by using domain ontology and information flow in enterprise information systems. To implement process-oriented and event-driven architecture, event collection and event buffering are assumed [2]. Juric proposed an extension of WSDL and BPEL for event-driven architecture. He presented an extended SOA/Web service platform to support event-driven architecture for business process integration. The platform was designed to integrate web service platform and event-driven process [3].
Meanwhile, there are many researchers who have investigated the adoption of service coordination for ubiquitous environment. Cai et al. proposed a novel intelligent service selection algorithm and application for ubiquitous web service environment [4]. Tsai and Wang presented computing coordination based on fuzzy group decision-making for web service oriented architecture by using computing power service [5]. Jung et al. suggested an ECA (Event-

Condition-Action) rule-based framework for decentralized coordination of ubiquitous web services. The framework included WS-ECA rule processing for efficient coordination using web services in ubiquitous computing environment [6].

3. Ubiquitous Process Coordination

In this section, we present the design of ubiquitous process coordination using RFID and USN. Figure 1 illustrates the scenario for the ubiquitous process coordination. Event sources in the scenario include information systems (i.e. CRM, ERP, and SCM) and ubiquitous devices (i.e. RFID, USN, and smart phones). The occurrences of logical events from information systems and physical events from the various devices are monitored and sent to the rule engine for event coordination.

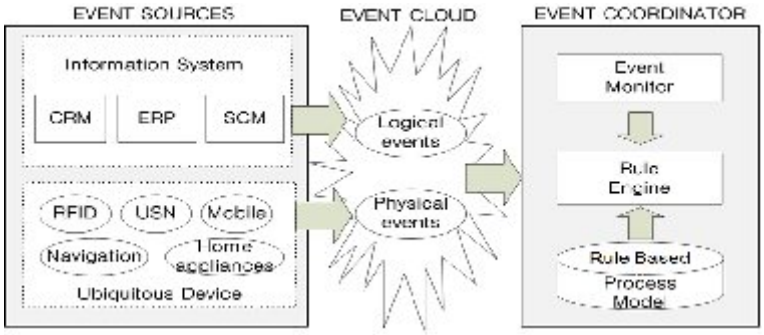

Figure 1: Framework of ubiquitous process coordination

3.1 RFID/USN Event Schema

In this section, we describe the schema of RFID and USN events. Table 1 shows the schema for RFID events: ID is the identification number of the tag which is captured by the RFID reader; Time describes the time point when the RFID event occurs; Reader contains the name of the RFID reader; and finally, counter is the reading counter of the RFID tag. Table 2 shows the schema for sensor events. For USN events, the attributes are mote_ID, d_type, ID, RTC, and ADC0. mote_ID is the sensor type number; d_type is the sensor device name; ID is the node identification number; RTC is date and time of the occurrence event; ADC0 gets input external data that include measurements of physical properties as temperature, humidity, light, acceleration, and other.

Table 1: Scheme of RFID events

Attribute	Description
ID	Identification number of RFID
Time	Occurrence time of the RFID event
Reader	Name of RFID reader
counter	Reading counter of the RFID tag.

Table 2: Scheme of sensor events

Attribute	Description
mote_ID	Sensor type number
d_type	Sensor device name
ID	Node identification number
RTC	Date/time of the occurrence event
ADC0	RFID door, gas, lamp, heater, acceleration, magnetic, sound, ultra – sonic, motion detector, vibration
etc	Internal temperature, external temperature, humidity, light

3.2 Rule Design for Service Coordination

Figure 2 illustrates the diagram of activity state transitions for the rule design to implement event-driven state transitions. In our research, activities of ubiquitous process coordination are divided to e-Service, which interacts with information systems, and u-Work, which communicates with ubiquitous devices. These activities enable process coordination on basis of ECA rules. The ECA rules are categorized into three kinds of rules: flow constraints rules, e-Service interaction rules, and u-Work detection rules. Flow constraints rules present control flow and message flow to execute business process. e-Service interaction rules invoke automated tasks and respond to the events of information system. u-Work detection rules recognize the start and end of each task by capturing physical events such as RFID/USN events.

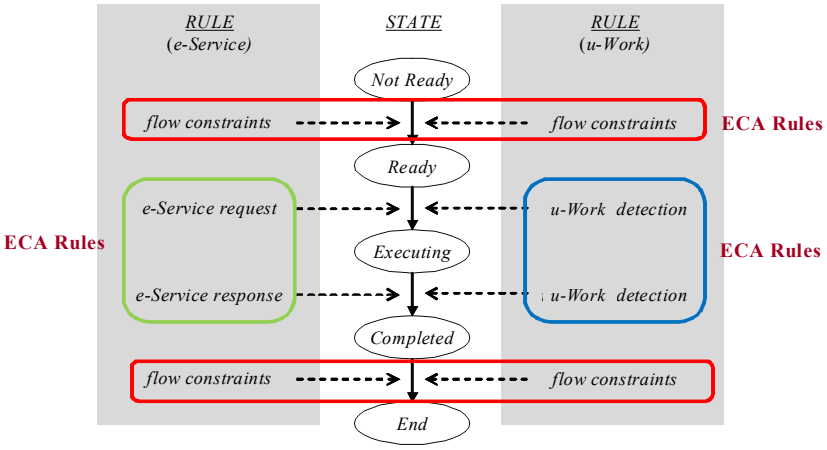

Figure 2: The diagram of activity state transitions and ECA rules

Table 3: A simple logistics process

Activity	Description	Type	Precedence	Antecedence	Join	Split
A1	unload items	u-Work		A2, A3		AND
A2	register to WMS	e-Service	A1	A4		
A3	check expiry date	e-Service	A1	A4		
A4	load items	u-Work	A2, A3		AND	

Table 3 presents a simple logistics process. This table consists of Flow Constraints Rules, e-Work interface Rule and u-Work. *Flow Constraints* Rule includes rule which presents process control flow and message flow for processing the precedence relations of work and the process progress. *E-Work* includes rules which can invoke and response automated work. *u-Work* Detection Rules include rules which identify start and end of work to act passively detecting ubiquitous event such as RFID/USN. Table 4 presents the examples of the expressing ECA rules written in Jess, which is a rule language [10]. In our proposed system, the three kinds of ECA rules for process coordination are transformed into Jess rules and are processed by the Jess engine.

Table 4: Expression of ECA rules in Jess rules

ECA Rule		Jess Rule
On IntEvt(activity.ready('A_1')) **AND** IntExt(sensor.detect('WH.unloading.itemAppear')) **Do** createIntEvt(activity.executing('A_1'))	⇒	(defrule A1_item_appear (and ?act <- (Activity {name == A_1 && state == ready}) (RFIDEvent (reader /WH/)(location /unloading/)(state /appear/))) => (modify ?act (state executing)))
On IntEvt(activity.executing('A_1')) **AND** IntExt(service.detect('WH.unloading.itemDisappear')) **Do** createIntEvt(activity.completed('A_1'))		(defrule A2_item_disappear (and ?act <- (Activity {name == A_1 && state == executing}) (RFIDEvent (reader /WH/)(location /unloading/)(state /disappear/))) =>(modify ?act (state completed)))

4. System Implementation

To illustrate the system design described in Section 3, we developed a ubiquitous process coordination system, named edUFlow (event-driven Ubiquitous Flow). Figure 4 shows the architecture of the proposed system. The system architecture consists of the *Ubiquitous Devices* Module, which gathers RFID and USN events and the *Process Coordinator*, which manages and controls the rules and process models.

In the system, passive RFID and active RFID transmit events of products and actors, respectively. The appearance and disappearance of products and actors can be recognized by the RFID readers. The Ubiquitous Sensor Network transmits sensor-detected events to the web service server [8-10]. Process Coordinator on the right side of the figure

manages and controls the events received from the ubiquitous devices. The detected events are first sent to the Event Monitor through adapters and web service servers; then, the filtered events are directed to the process engine to trigger proper rules in the process model. The process engine can invoke web services of external information systems and activate the sensors to control the ubiquitous environment.

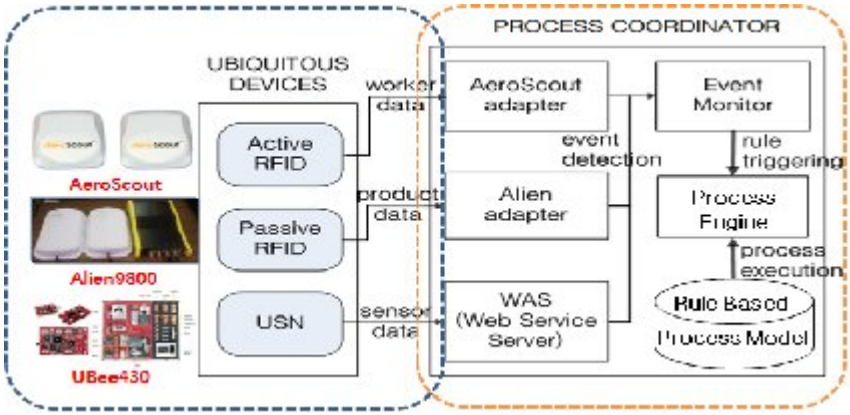

Figure 4: ubiquitous process coordination system architecture

4.1 System Design

The class diagram of the edUFlow system illustrates in Figure 5. The classes in the diagram are largely categorized into four groups: service invocation (logical event), ubiquitous devices (physical events), event monitor, and process engine. InvokeService, ServiceRequest are classes for service invocation (logical events). RFIDEvent and SensorEvent are classes for ubiquitous devices (physical events). Thread_RFID and RFIDMessageListener are classes for event monitor. Process engine consists of RFIDMessageListener, ProcessMonitorUI and ProcessEngine classes. The dynamics of the process is as follows: RFIDMessageListener class exchanges data in XML format, and then it transmits data to the ProcessEngine. The USNWebservice invokes the ProcessEngine to obtain the necessary data when an event occurs in real-time. The ProcessMonitorUI displays the events originated on the ProcessEngine on the computer screen. The ProcessEngine modifies and saves events using Jess Rule. This class has the member variables which are the class objects such as Rete, WorkingMemoryMaker, and ProcessMonitorUI. The class invokes the various objects, this class presents user interface.

4.2 Rule Language Implementation

Figure 6 illustrates outline of jess rules for process coordination. Fundamental construction of the Jess includes Jess rule language includes six fundamental constructions: *Java Binding* declares class of external service, RFID, USN etc.. *Process Components template* is a declarative part for rule based design of ubiquitous process by using the Jess rule language. *Process Enactment* starts and ends the ubiquitous process, and also Process Enactment defines process control order of stoppage and resumption etc. by function of the Jess language. *Flow Constraints* are the rule which presents antecedence for the transition of the process execution and the activities. *e-Service request/response* describes the rule which requests and responses the services from the information system. *u-Work* is the rule for the implementation of the real-time work and monitoring.

4.3 User Interface Implementation

In this section, we describe how users can edit event patterns and event-driven rules and monitor the progress of event occurrence and rule processing in user interface. edUFlow includes three main user interfaces: *Ubiquitous Event Pattern, Ubiquitous Event Filter* and *Ubiquitous Process Manager* (see Figures 7 and 8). Even though the screen capture of *Ubiquitous Event Pattern Editor* is omitted, because of the limit of space, users can define event patterns based on the sensor types and arithmetic operators. In *Ubiquitous Event Filter*, users can define event patterns for RFID and USN events, and monitor the simple and complex events which are filtered by the event patterns. The left part of Figure 7 shows raw events which are generated from USN and RFID, and the right of the figure shows the filtered events by the user-defined pattern. In *Ubiquitous Process Manager*, users can edit Jess rules for process coordination and monitor the log of events and process execution. As shown in Figure 8, users can select and edit a Jess rule file which contains the rules, which are briefly introduced in Figure 6. The interface also shows the list of events which influence the rules and the log of rule processing in the right part of the screen.

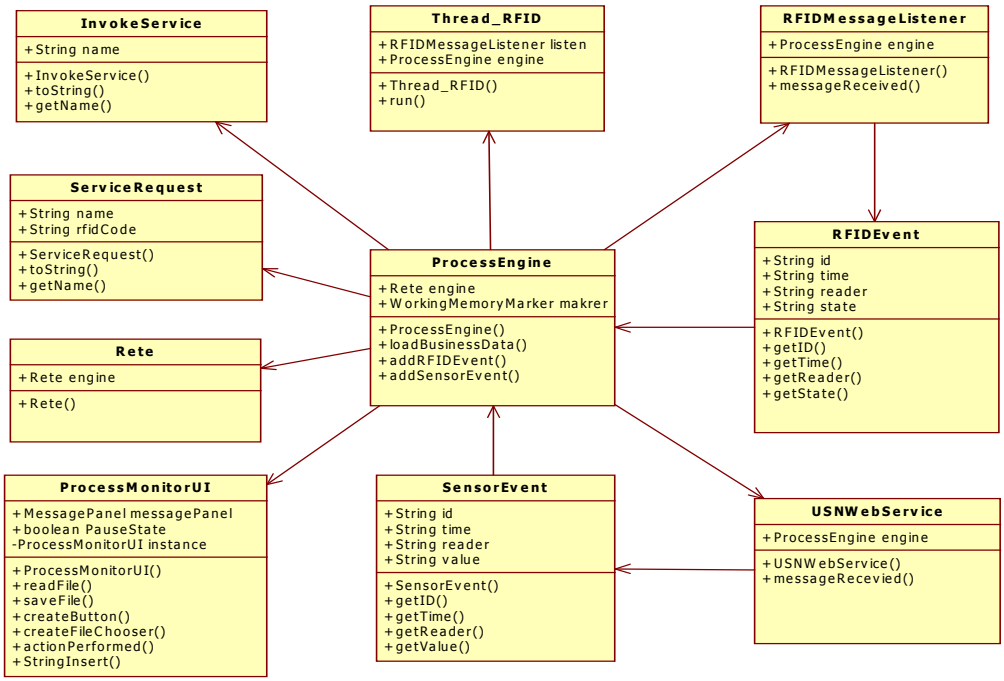

Figure 5: class diagram of the edUFlow system

//Java Binding
(import org.khu.bpm.bre.model.*)
(deftemplate InvokeService (declare (from-class InvokeService)))
(deftemplate RFIDEvent (declare (from-class RFIDEvent)))
(deftemplate ServiceRequest (declare (from-class ServiceRequest)))

//component
(deftemplate Process
"generic process"
(slot name)
(slot state)
(slot firstActName)
(slot finalActName))
(deftemplate Activity
"generic activity"
(slot name)
(slot state))
(deftemplate Service
"generic service to invoke"
(slot name)
(slot type))

(deftemplate Event
(slot id)
(slot rfidCode))

< abbreviated>

//Process Enactment
(deffunction
initiateProcess(?procName ?rfidCode)
(assert (Process (name ?procName)
(rfidCode ?rfidCode)
(state running) (firstActName A1)))
(assert (Activity
(name A1)
(state completed))))

(defrule A1_service_request
?svc <- (ServiceRequest)
=>
(initiateProcess
testProcess ?svc.rfidCode))

(defrule A2_rfid_appear
(and ?act <- (Activity
{name == A2 && state == ready})
(RFIDEvent (id /R001/) (state
/appear/)))
=>
(modify ?act (state executing)))

< abbreviated>

//Flow Constraint
(defrule A1_end
?act <- (Activity {name == A1
&& state == completed})
=>
(and (assert (Event (id E_1_3)))
(assert (Event (id E_1_2)))
)
(printout t "//E_1_3, E_1_2 occurrence:" crlf)
)
(defrule A2_start
(Event (id /E_1_2/))
=>
(assert (Activity (name A2) (state ready)))
(printout t "//A2 start:" crlf)
)
< abbreviated>

//e-Service Request/Response
(defrule Al_service_request
?svc <- (ServiceRequest)
=>
(initiateProcess testProcess ?svc.rfidCode))

//u-Work Detection
(defrule A2_rfid_appear
(and ?act <-(Activity
{name == A2 && state == ready})
(RFIDEvent (id/R001/) (state /appear/)))
=>
(modify ?act(state executing)))

Figure 6: Outline of Jess rules for process coordination

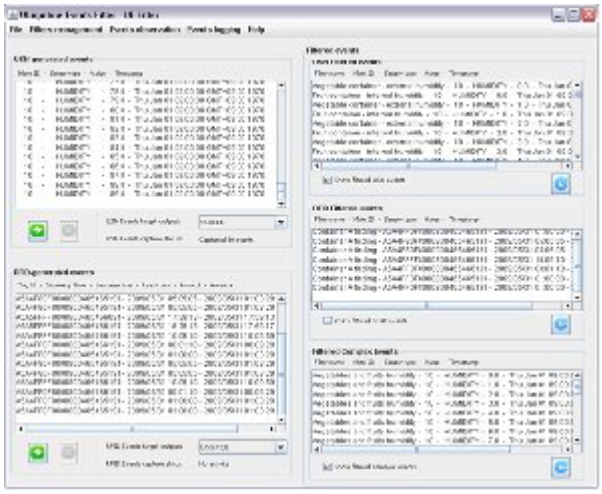

Figure 7: Ubiquitous Event Filter

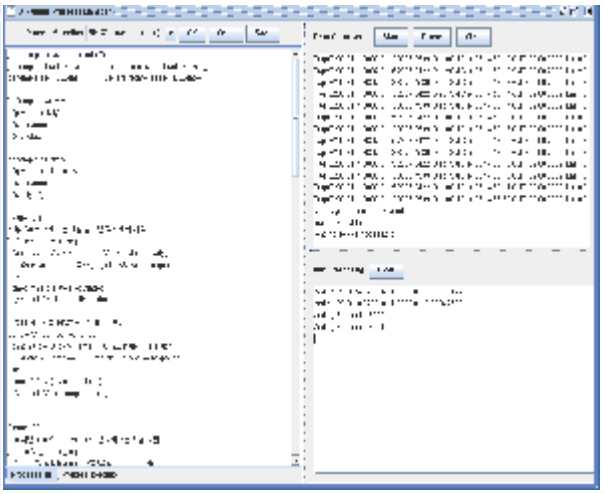

Figure 8: Ubiquitous Process Manager

5. Conclusion

A variety of devices and computing systems such as RFID, USN and mobile devices generate significant events in modern information systems. In this paper, we presented the edUFlow system which integrates ubiquitous environments and business processes using the process coordination. The goal of the system is to illustrate how to integrate real-time events to process coordination systems by adopting complex event processing and event-driven rule technologies. The system architecture has the following benefits. First, the system applies event web technology based on web protocol for occurrence real-time events; the system can execute distributed processes which operate various events multiply. Second, the system can monitor the various events on business integration environment. Third, the system can manage and control occurrence complex events, and can confirm the filtering data required from the user.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant (No. 2010-0070475) and the Korea Research Foundation Grant (KRF-2008-331-D00709) funded by the Korea government (MEST).

References

1. Ammon, R. V. et al., 2010, “On Measuring Business Value of CEP-Applications,” ACM International Conference on Distributed Event-Based Systems.
2. Henglein, F., Larsena, K. F., Simonsena, J. G., and Stefansena, C., 2009, “POETS: Process-Oriented Event-Drivem Transaction System,” Journal of Logic and Algebraic Programming (JLAP), 53(1), 381-401.
3. Juric, M., 2010, “WSDL and BPEL extensions for Event Driven Architecture,” Information and Software Technology, 52(7), 1023-1043.
4. Cai, H., Hu, H., Lü, C., and Cao, Q., 2009, “A novel intelligent service selection algorithm and application for ubiquitous web services environment,” Expert Systems with Applications, 36(2), 2200-2212.
5. Tsai, M.-J., and Wang, C.-S., 2008, “A computing coordination based fuzzy group decision-making (CC-FGDM) for web service oriented architecture,” Expert Systems with Applications, 34(4), 2921-2936.
6. Jung, J.-Y., Park, J., Han, S.-K., and Lee, K., 2007, “An ECA-based framework for decentralized coordination of ubiquitous web services,” Information and Software Technology, 49(11-12), 1141-1161.
7. Jess, the Rule Engine for the Java Platform, http://www.jessrules.com.
8. OASIS, 2009, Web Services Coordination (WS-Coordination) Version 1.2, http://docs.oasis-open.org/ws-ex/wscoor/2006/06.
9. OASIS, 2009, Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2, http://docs.oasis-open.org/ws-tx/wsat/2006/06.
10. OASIS, 2009, Web Services Context (WS-Context) Version 1.0, http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html.