능동형 규칙 기반 유비쿼터스 프로세스 설계의 워크플로우 패턴 분석

Pattern-based Analysis of Ubiquitous Process Design Based on Active Rules

정재윤(Jae-Yoon Jung)*, 박종헌(Jonghun Park)**

초 록

프로세스 설계에는 Petri-net, EPC, UML Activity Diagram 등 다양한 모델 기법이 사용되고 있다. 업무 프로세스 내에 복잡한 업무 규칙이 다수 결합되어 있거나, 유비쿼터스 컴퓨팅과 같이 분산 환경에서 복합한 상호운용 규칙이 산재되어 있는 경우에는 정형적인 프로세스 모델링 기법이 비효율적이므로 규칙 기반의 분산 프로세스 설계를 사용할 수 있다. 본 연구는 유비쿼터스 환경에서의 규칙 기반 프로세스 설계 방법을 분석한다. 특히, 이벤트-조건-액션(ECA) 형태의 능동형 규칙을 이용한 프로세스 설계를 대상으로 분석하며, ECA 스키마는 웹 서비스 기술을 위해 개발된 WS-ECA 언어를 사용한다. ECA 규칙을 이용한 여러 가지 프로세스 통제 흐름 표현 방법을 분석하기 위하여 워크플로우 패턴을 활용하였다. 워크플로우 패턴에 따라 ECA 규칙 기반 프로세스 표현법을 제공함으로써 규칙 기반 프로세스를 설계하기 위한 가이드라인을 제시하였다는 데 연구의 의의가 있다.

ABSTRACT

Many process modeling techniques, such as Petri-net, UML Activity Diagram, and EPC, are used to design process models. In this paper, we analyze ubiquitous process design based on workflow patterns. In particular, we focus on process design using active rules that have the form of Event-Condition-Action, and deal with the WS-ECA language, which was devised for ubiquitous web services coordination. We first check whether workflow patterns can be designed with ECA rules, and we then provide WS-ECA representations for ECA rules of the patterns. The contribution of this paper is that ECA rule-based process models were presented based on workflow patterns and they can be a guideline for ubiquitous process modeling.

키워드 : 유비쿼터스 프로세스 설계, 능동형 규칙, WS-ECA, 워크플로우 패턴
Ubiquitous Process Modeling, Active Rules, WS-ECA, Workflow Patterns

---

이 논문은 경희대학교의 지원을 받아 수행되었음.
* 경희대학교 산업경영공학과 전임강사, 산학협력기술연구원
** 서울대학교 산업공학과 부교수
2008년 10월 05일 접수, 2009년 02월 04일 심사완료 후 2009년 02월 09일 게재확정.

1. 서 론

프로세스 설계 및 관리는 비즈니스 프로세스의 설계 및 분석(예 : transactional workflow, 6시그마), 기업간 거래 표준 프로세스의 명세(예 : ebXML, RosettaNet), 과학적 분석 프로세스(예 : scientific workflow, grid workflow) 등 다양한 분야에서 활용되고 있다. 특히, 비즈니스 프로세스나 기업간 거래 프로세스와 같이 정형화된 업무 과정의 설계로는 Petri-net [1], UML Activity Diagram[5], EPC(Event-Driven Process Chain)[2]와 같은 모델링 방법이 사용된다. 그러나 복잡하고 비정형적인 업무 규칙을 포함하는 정보시스템의 통합이나, 분산 환경의 임의적인 메시지 교환이 필요한 유비쿼터스 시스템 통합에서는 전체적인 프로세스의 설계과 관리가 용이하지 않기 때문에, 분산 환경에서 실행되고 독립적으로 실행될 수 있는 규칙 기반의 프로세스 설계 및 관리 방식이 효과적이다[7].

본 연구는 유비쿼터스 서비스를 통합하기 위한 규칙 기반 프로세스 설계 방법을 분석 대상으로 한다. 특히, 규칙 기반 프로세스 설계를 통하여 각 워크플로우 패턴을 설계하는 것이 가능한지 살펴보고, ECA 규칙 기반 웹 서비스 언어인 WS-ECA(ECA rule description language for Web Services) 언어로 표현하는 방법을 설명한다. van der Aalst, ter Hofstede 등에 의해서 개발된 워크플로우 패턴[3]은 프로세스 설계에서 사용되는 기본적인 통제 흐름(control-flow)을 추출한 것으로, BPEL4WS, UML Activity Diagram, EPC, BPMN 등 프로세스 설계 표준 언어의 프로세스 표현 능력을 분석하거나[9, 13, 14, 16], IBM WebSphere, Oracle BPEL, Staffware 등의 프로세스 설계 시스템의 모델링 능력을 평가하는 데[17, 15] 활용되고 있다. 이러한 측면에서 규칙 기반의 유비쿼터스 프로세스 설계를 위한 WS-ECA 언어를 분석하는 데에도 유용할 것으로 판단된다.

WS-ECA 언어는 유비쿼터스 환경에서 ECA 규칙에 근거하여 서비스 코디네이션을 위하여 제시되었으며, 이벤트 기반의 규칙 표현을 기반으로 전형적인 능동형 규칙을 표현하고 있는 웹 서비스 응용 언어이다[6, 8]. 규칙 기반의 프로세스 설계 및 실행은 전통적인 중앙집중식 프로세스 설계 및 실행에 비하여 확장성 있는(extensible) 프로세스 설계가 가능하고, 독립적이고 분권화된(decentralized) 통제 및 관리가 가능하며, 분산 환경에서의 경량적(lightweight) 프로세스 구현이 가능하다[7]. WS-ECA에 기반한 프로세스 설계 및 실행을 위한 구조 및 아키텍처는 관련 연구를 참조하기 바란다[6, 7].

본 논문은 제 2절에서 WS-ECA 스키마에 대하여 소개하고, 제 3절에서는 ECA 표현을 위한 BPMN 표기법을 설명한다. BPMN 표기는 WS-ECA의 XML 표현을 단순화하여 표현하기 위하여 도입되었다. 제 4절에서는 워크플로우 패턴에 따라 규칙 기반 프로세스 설계 가능성을 분석하였으며, 제 5절에서는 표현된 패턴들을 WS-ECA로 변환하는 방법을 설명하였다. 제 6절에서는 헬스케어 프로세스 설계를 패턴에 근거하여 WS-ECA 언어로 표현하는 예제를 제시하였다. 제 7절에서 결론 및 추후연구를 제시하였다.

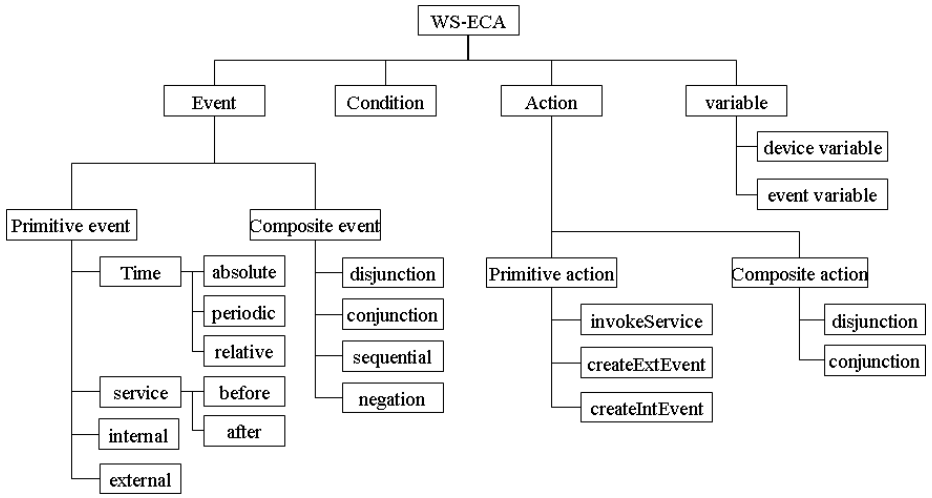
2. 웹 서비스 규칙 언어 : WS-ECA

웹 서비스 환경에서 활용할 수 있는 규칙 언어로는 RuleML[11], WRL(Web Rule Language)[12], WSML-Rule(Web Service Markup Language)[4], WS-ECA[8] 등이 있는데, 이 중 WS-ECA는 이벤트-조건-액션(Event-Condition-Action) 형식으로 능동형 규칙을 웹 서비스 환경에 적용하기 위하여 개발된 언어이다. WS-ECA는 유비쿼터스 웹 서비스 환경에서 기기들이 WS-Eventing을 이용하여 상호협력적인 규칙 처리를 수행하기 위하여, <그림 1>과 같이 이벤트, 조건, 액션, 변수로 구성되어 있는데, 웹 서비스를 지원하기 위한 네 가지 기본 이벤트를 지원한다[6].

WS-ECA는 <그림 1>과 같이 크게 이벤트, 조건, 액션, 변수로 구성되는데, 특히 이벤트는 유비쿼터스 환경의 웹 서비스를 지원하기 위하여 시간 이벤트(time event), 서비스 이벤트(service event), 내부 이벤트(internal event), 외부 이벤트(external event)라는 네 가지 형태의 기본 이벤트를 사용하여 서비스를 구성하고 결합할 수 있다. 내부 이벤트는 하위 프로세스 내부에서 통제 흐름(control flow)을 표현하기 위하여 사용되고, 외부 이벤트는 외부 프로세스 간의 메시지 흐름(message flow)을 표현하기 위하여 사용된다. 또한, AND, XOR, OR의 분기 및 병합과 같은 프로세스 구조적 제어는 기본 이벤트의 조합으로 구성되는 복합 이벤트(composite event)와, 규칙 내의 조건(condition)을 활용하여 표현할 수 있다.

3. ECA 규칙의 BPMN 표기

본 연구에서는 ECA 규칙 기반의 프로세스 모델을 표현하기 위하여, 비즈니스 프로세스 표준 표기법인 BPMN(Business Process

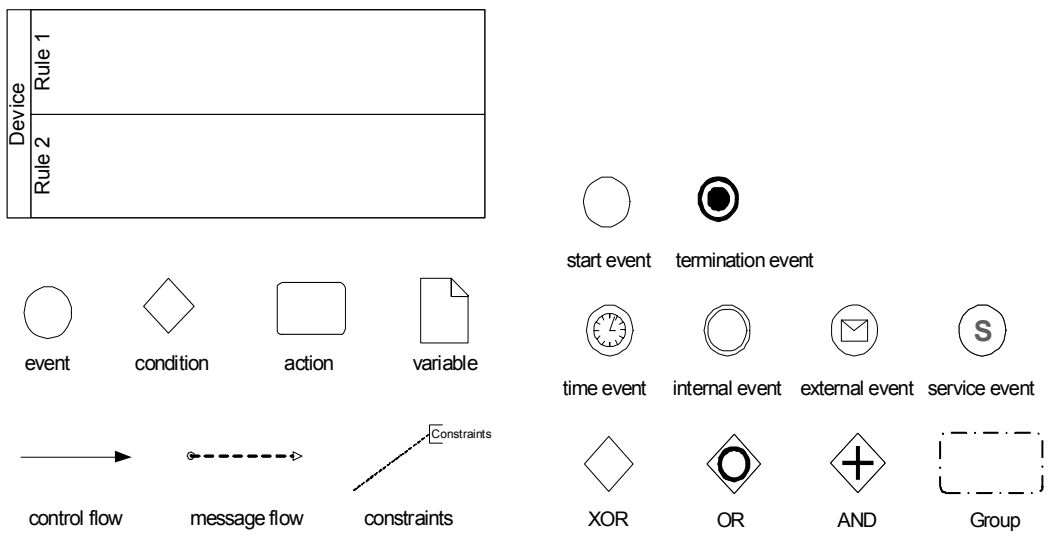


〈그림 1〉 이벤트 기반 규칙 언어(WS-ECA)의 기본 구조

Modeling Notations)[10]을 이용한다. BPMN을 사용하여 규칙 언어를 설명하는 이유는 WS-XML 언어로 직접 표현하는 것보다 간결하고 시각적으로도 이해하기 쉽기 때문이다[13]. BPMN은 비즈니스 프로세스를 표현하는 데 필요한 방대한 표기들을 제공하고 있으나, 본 연구의 대상인 규칙 기반 설계 언어인 WS-ECA로 직접 변환되기 위해서는 <그림 2>에서 기술된 일부분만을 사용해야 한다.

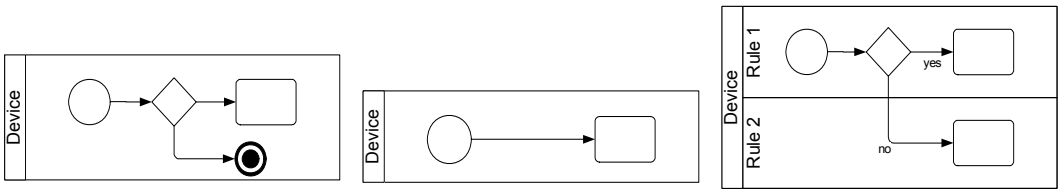
유비쿼터스 프로세스 실행에 참여하는 하나의 시스템은 다수의 규칙 언어, 즉 다수의 하위 프로세스를 포함할 수 있는데, 시스템과 프로세스는 각각 Pool과 Lane으로 표현된다. <그림 2(a)> 상단의 외부 직사각형인 Pool은 하나의 독립적인 시스템에 해당되고, Pool 내의 각 Lane은 하나의 규칙에 해당된다. WS-ECA 언어에서는 Pool이 하나의 WS-ECA 문서로 표현되며, 각 Lane은 WS-ECA 문서 내의 <rule> 엘리먼트 하나로 기술된다.

다음으로 ECA 규칙의 주요 구성요소인 이벤트(Event), 조건(Condition), 액션(Action)은 BPMN 기호에서 <그림 2(a)>와 같이 각각 원, 마름모, 둥근 사각형으로 표기된다. 나아가 WS-ECA에서는 <그림 1>과 같이 조건 및 데이터를 표현하기 위하여 Variable을 사용하는데, 이는 BPMN의 문서기호로 표기된다. <그림 2(a)>의 하단과 같이 구성요소 간의 흐름은 통제 흐름(control flow, 실선으로 표기)과 메시지 흐름(message flow, 점선으로 표기)으로 분류되는데, 통제 흐름이란 동일 시스템 내에서 일어나는 이벤트의 발생, 조건의 만족, 액션의 수행, 이벤트의 전달을 표현할 수 있으며, 메시지 흐름이란 서로 다른 시스템 간에 교환되는 이벤트 전



(a) 기본적 기호 (b) 확장적 표기

〈그림 2〉 규칙 기반 프로세스 표현에 사용되는 BPMN 기호들



(a) When-if-then form (b) When-then form (true condition) (c) When-if-then-else form

〈그림 3〉 기본적인 WS-ECA 규칙의 BPMN 표기

달을 표현한다. 조건부(Condition)에서 제약 조건을 나타내기 위해서는 BPMN의 constraints 표기를 이용할 수 있다.

이벤트의 유형을 분류해보면, 시작과 종료 이벤트와 WS-ECA 규칙에서 사용되는 네 가지 기본 이벤트 유형, 즉 time event, service event, internal event, external event는 <그림 2(b)>와 같이 표기된다. 또한, 마름모 형태의 조건은 XOR, OR, AND 유형의 분기와 병합을 마름모 내에 표기할 수 있으며, 일점쇄선을 이용한 둥근 직사각형으로 표기되는Group 표시를 이용하여 복합 이벤트 또는 복합 액션(composite action)을 표현할 수 있다.

ECA 규칙의 일반적인 형태는 <그림 3(a)>와 같이 표현된다. 즉, 하나의 이벤트가 발생

```
<ECARule name = "basic_rules"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    xmlns = "http://di.snu.ac.kr/2005/eca/">
    <events>
        <intEvent name="evt_a"/>
    </events>
    <actions>
        <invoke name = "act_a" service="eg:S1"/>
        <invoke name = "act_b" service="eg:S2"/>
    </actions>
    <rules>
        <rule name = "rule_a">
            <event name = "evt_a"/>
            <condition expression = "cond = true"/>
            <action name = "act_a"/>
      </rule>
      <rule name = "rule_b">
            <event name = "evt_a"/>
            <condition expression = "cond = false"/>
            <action name = "act_b"/>
      </rule>
   </rules>
</ECARule>
```

〈그림 4〉 기본적인 WS-ECA 규칙 문서 예제

하면 규칙 검사가 실행되고, 조건을 만족하지 않으면 종료 이벤트를 발생하며, 조건이 만족되면 액션을 실행하는 과정이다. 조건 단계가 항상 참인 경우는 <그림 3(b)>와 같이 약식으로 표현될 수 있다. <그림 3(c)>는 조건의 만족 유무에 따라서 서로 다른 액션이 실행되는 경우에 두 개의 규칙으로 나누어 표현하는 예를 보여준다. 이 두 가지 규칙을 하나의 시스템에 설계한다고 가정하면 <그림 4>와 같은 WS-ECA 문서로 기술할 수 있다.

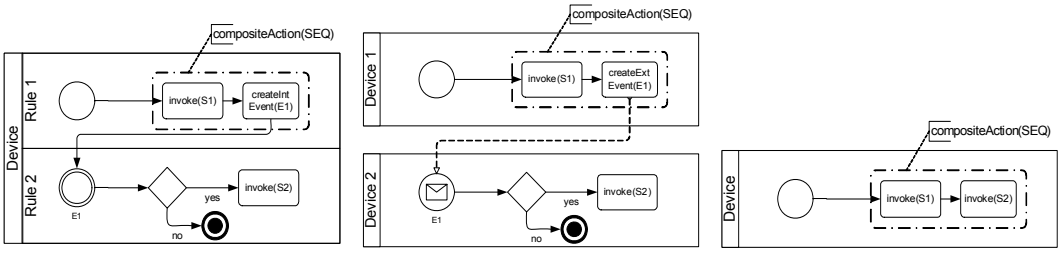
4. 규칙 기반 서비스 설계의 프로세스 패턴 분석

본 절에서는 ECA 규칙의 BPMN 표기를 사용하여 여러 가지 프로세스 패턴을 표현하는 방법과 가능성을 검증한다. 프로세스 패턴은 비즈니스 프로세스 기술 언어를 분석하기 위하여 널리 사용되고 있으며, 본 연구에서 제시된 규칙 기반의 프로세스의 표현력을 분석하는 데에도 유용한 도구로 사용될 수 있다. 워크플로우 패턴은 총 여섯 가지 카테고리로 나누어지는데[3], 각 카테고리의 패턴이 유비쿼터스 프로세스 설계에서 가지는 의미를 살펴보고 WS-ECA 규칙 표기로 어떻게 표현할 수 있는지 기술한다.

4.1 기본 통제 패턴

첫 번째 카테고리는 기본적 통제 패턴(basic control flow patterns)으로서 직렬화(Sequence : 패턴 1), 병렬적 분기(Parallel Split : 패턴 2), 동기화(Synchronization, 패턴 3), 배타적 분기(Exclusive Choice : 패턴 4), 단순 병합(Simple Merge : 패턴 5)을 포함한다.

직렬화(패턴 1)는 순차적 통제 패턴으로 다른 두 개의 서비스를 순서대로 호출하는 역할을 한다. WS-ECA의 Action에는 복합 액션(composite action)이 존재하여 순차적 액션(sequential action) 또는 동시 액션(conjunction action)을 표현할 수 있다. <그림 5(a)>는 복합 액션을 이용하여 두 개의 액션을 한 시스템 내에서 순차적으로 호출하는 예를 보여준다. 한편 조건 비교를 수행한 후에 두 번째 서비스 호출을 수행하는 경우에는 eventing을 통하여 둘 이상의 서비스를



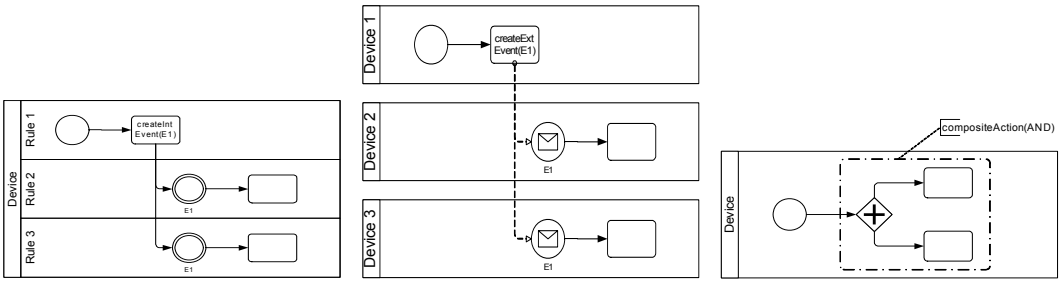
(a) 내부 이벤트 활용 (b) 외부 이벤트 활용 (c) 복합 액션("SEQ") 활용

〈그림 5〉 직렬화(패턴 1)

순차적으로 실행할 수 있는데, 동일 시스템 내에서 전달되는 내부 이벤트 생성(createIntEvent) 액션과, 다른 시스템에 메시지를 전달하는 외부 이벤트 생성(createExtEvent) 액션을 사용하여 <그림 5(b)>와 <그림 5(c)>로 표현할 수 있다. 참고로 각 액션에 의해 생성된 흐름은 통제 흐름(control flow)과 메시지 흐름(message flow)으로 각각 실선과 점선으로 표기된다.

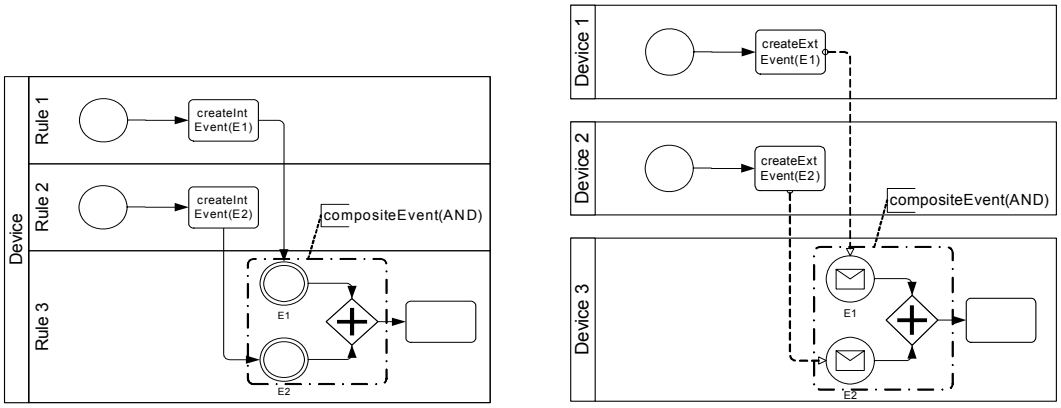
병렬적 분기(패턴 2)는 병렬 분기 패턴으로 하나의 이벤트가 다수의 서비스에 영향을 주는 패턴이다. 패턴 1과 같이 복합 액션을 직접 사용하는 방법, 내부 시스템과 외부 시스템에 이벤트를 통해 전달하는 경우가 있다. <그림 6(a)>는 AND 형태의 복합 액션(conjunction action)을 통하여 병렬 수행하는 예이고, <그림 6(b)>와 <그림 6(c)>는 각각 내부 이벤트와 외부 이벤트를 통해 병렬 분기를 수행하는 예이다. 나아가, WS-ECA에서는 조건 검사 단계에서 조건별 분기를 표현할 수 있다.

동기화(패턴 3)는 동기화 패턴으로서 둘



(a) 내부 이벤트 활용 (b) 외부 이벤트 활용 (c) 복합 액션(“AND”) 활용

〈그림 6〉 병렬적 분기(패턴 2)



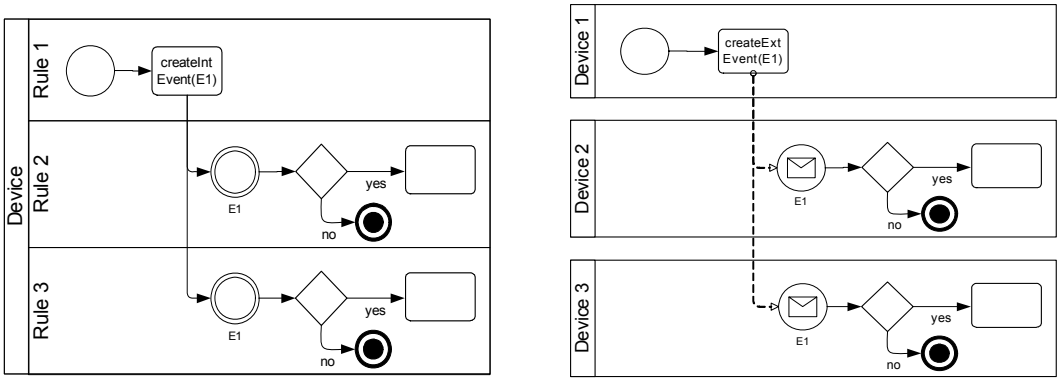
(a) 내부 이벤트 활용 (b) 외부 이벤트 활용

〈그림 7〉 동기화(패턴 3)

이상의 서비스가 모두 종료되어야만 다음 서비스를 개시하는 패턴이다. 이 패턴도 내부 시스템과 외부 시스템의 경우로 나누어 <그림 7(a)>와 <그림 7(b)>로 표현할 수 있다.

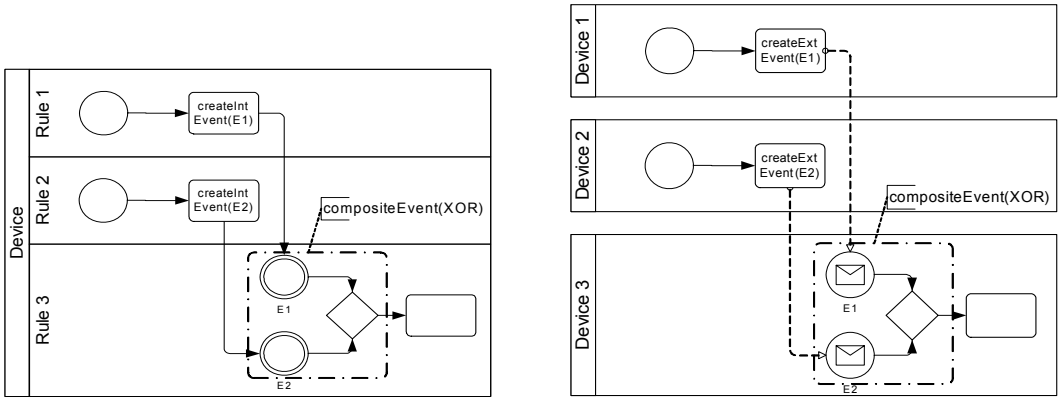
배타적 분기(패턴 4)는 배타적 선택 분기 패턴으로 조건에 따라 하나의 서비스만 실행하는 패턴이다. WS-ECA에서는 병렬 분기 이후에 각각의 조건 검사를 통하여 검사할 수 있다. 특히, 이때의 조건들이 배타적 조건이면 배타적 분기(패턴 4)이며, 교차적 조건이면 다중 분기(패턴 6)에 해당된다.

단순 병합(패턴 5)은 단순 병합 패턴으로 두 가지 이상의 이벤트 중 어느 하나라도 발생하면 항상 반응하는 서비스 패턴을 의미한다. 동기화 패턴과 마찬가지로 다수의 이벤트에 대하여 반응하는 규칙을 생성하되, OR 형태의 복합 이벤트를 사용하여 표현할 수 있다.



(a) 내부 이벤트 활용 (b) 외부 이벤트 활용

〈그림 8〉 배타적 분기(패턴 4) 또는 다중 분기(패턴 6)



(a) 내부 이벤트 활용 (b) 외부 이벤트 활용

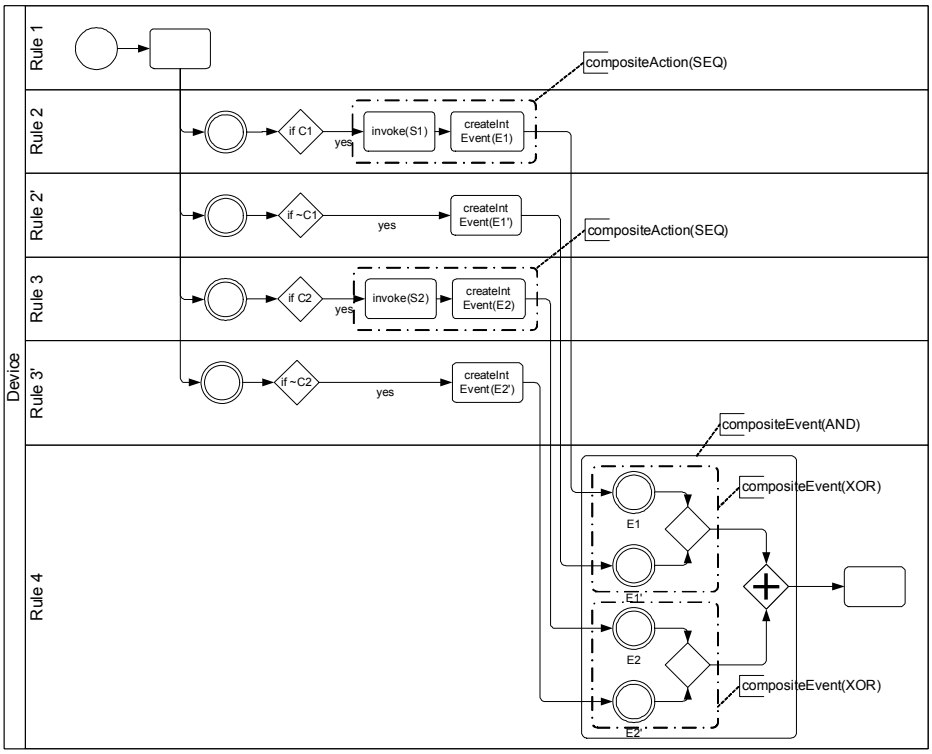
〈그림 9〉 단순 병합(패턴 5)

4.2 고급 분기 및 동기화 패턴

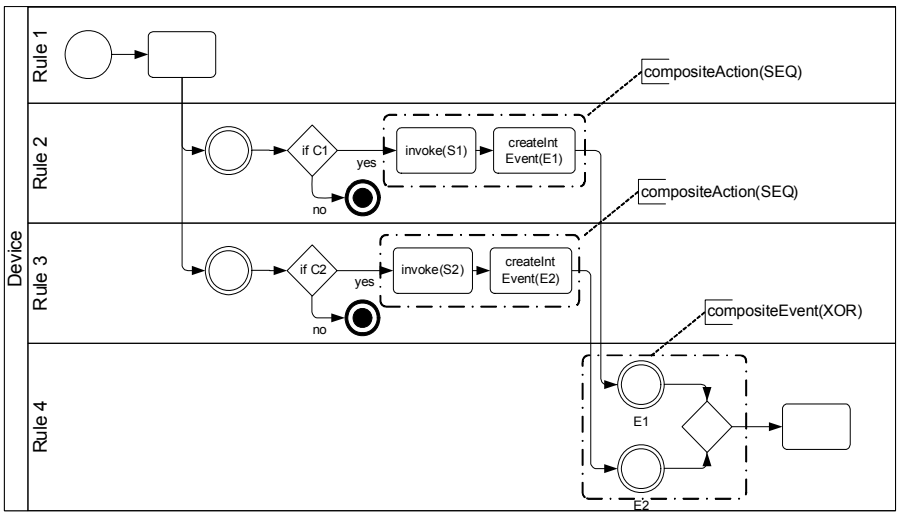
프로세스 패턴의 두 번째 카테고리는 고급 분기 및 동기화 패턴(Advanced branching and synchronization patterns)으로 다중 분기(Multiple Choice : 패턴 6), 동기적 병합(Synchronizing Merge : 패턴 7), 다중 병합(Multiple Merge : 패턴 8), 식별적 병합(Discriminator : 패턴 9)를 포함한다.

다중 분기(패턴 6)는 다중 선택 분기로서 표현 방법은 패턴 4에서 이미 설명되었듯이, 두 규칙의 조건이 동시에 만족할 수 있는 경우가 있도록 교차적 조건으로 정의되었다면, 동일한 이벤트에 대하여 둘 이상의 활동을 수행할 수 있으므로 다중 분기 패턴에 해당된다.

동기적 병합(패턴 7)은 동기화 병합 패턴으로 조건에 따라 발생 가능성이 불확실한 둘 이상의 서비스를 동기화시키는 패턴이다. <그림 10>의 예에서 Rule 2의 서비스 S1은 조건 C1이 만족해야만 실행되고, Rule 3의 서비스 S2도 조건 C2가 만족해야만 실행된다. 그런데, Rule 4의 액션은 두 서비스가 만약 실행되는 조건이라면 둘다 실행이 완료될 때까지 기다린 후에 액션을 시작해야 하며, 어느 하나만 실행되는 조건이라면 하나만 종료되면 즉각 액션을 시작시켜야 한다. 이러한 동기적 병합 패턴은 조건이 만족하지 않는지를 판정할 수 있도록 Rule 2'와 Rule 3'를 추가로 기술한 후에, Rule 2와 Rule 2' 중



〈그림 10〉 동기적 병합(패턴 7)



〈그림 11〉 다중 병합(패턴 8)

하나가 실행 완료되고(Rule 4의 첫 번째 XOR 복합 이벤트) 동시에 Rule 3와 Rule 3'중 하나가 실행 완료되는(Rule 4의 두 번째 XOR 복합 이벤트) 두 가지 복합 이벤트가 모두 실행되는 AND 복합 이벤트를 중복적으로 설계함으로써 표현할 수 있다.

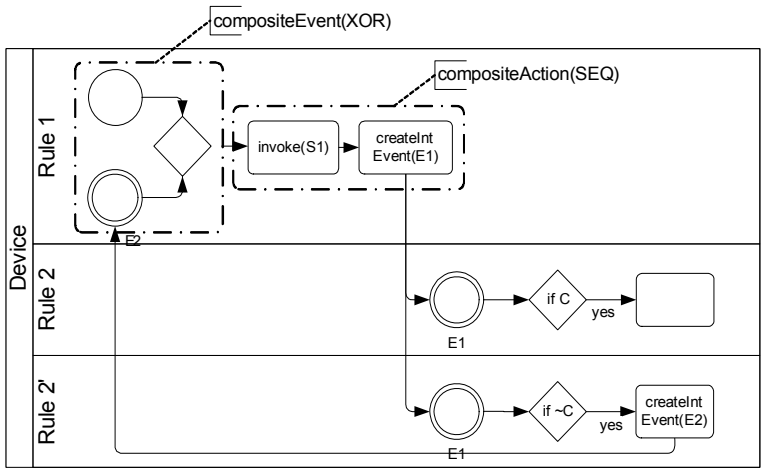
다중 병합(패턴 8)은 다중 병합 패턴으로 하나 이상의 불확실한 상황을 동기화시켜 하나의 흐름으로 통합하는 것이 아니라, 개별 인스턴스로 취급하여 다수의 흐름으로 실행하는 패턴이다. 이는 병렬 분기를 XOR 복합 이벤트로 병합함으로써 쉽게 표현된다.

식별적 병합(패턴 9)은 하나 이상의 불확실한 상황에 대하여, 단 한번만 서비스하고 이후에 일어나는 상황은 무시하는 패턴이다. 이는 복합 이벤트에서 처음 발생한 이벤트는 시작시키고 이후에 발생한 이벤트는 무시하기 위하여 상태를 기억해야 하야만 처리할 수 있는데, 상태를 가정하지 않는 WS-ECA 규칙으로는 직접 표현하기 어려운 패턴이다. BPEL이나 WSCI와 같이 상태를 기억하는 대부분의 프로세스 실행 언어들조차 반영하고 있지 않는 특수한 패턴이다.

4.3 구조적 패턴

세 번째 카테고리는 구조적 패턴(Structural patterns)으로 임의적 순환(Arbitrary Cycles : 패턴 10), 암시적 종료(Implicit Termination : 패턴 11)를 포함한다. 임의적 순환(패턴 10)은 특정 구간을 순환하는 패턴으로 <그림 12>에 기술된 것과 같이 순환을 위한 보조적 규칙을 배타적 조건을 사용하여 정의함으로써 표현가능하다. 즉, 서비스 S1은 조건 C를 만족시키지 못하면, Rule 2'에 의하여 Rule 1이 다시 시작되어, 조건 C를 만족시킬 때까지 반복된다.

한편, 암시적 종료(패턴 11)는 사용자가 개입하여 업무의 실행과 종료를 결정하는 패턴이다. 본 연구에서 가정하는 유비쿼터스

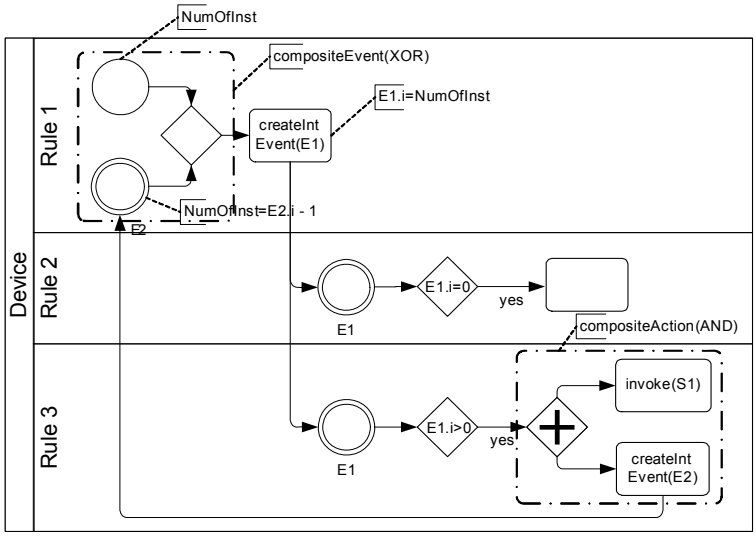


〈그림 12〉 임의적 순환(패턴 10)

규칙 프로세스는 사용자 개입을 가정하지 않고 정해진 규칙에 기반하여 실행하는 것을 가정하고 있어서 암시적 종료 패턴은 고려하지 않는다. 만약 사용자 개입이 필요하다면, 시스템을 통하여 규칙을 트리거 할 수 있는 이벤트를 간접적으로 발생시키는 방법으로 접근이 가능하다.

4.4 다중 인스턴스(MI) 패턴

네 번째 카테고리는 다중 인스턴스(Multiple Instances : MI) 패턴으로 비동기적 다중 인스턴스(MI without Synchronization : 패턴 12), 설계시 결정 다중 인스턴스(MI with a Priori Design Time Knowledge : 패턴 13),



〈그림 13〉 비동기적 다중 인스턴스(패턴 12)

실행시 결정 다중 인스턴스(MI with a Priori Runtime Knowledge : 패턴 14), 비결정적 다중 인스턴스(MI without a Priori Runtime Knowledge : 패턴 15)를 포함한다.

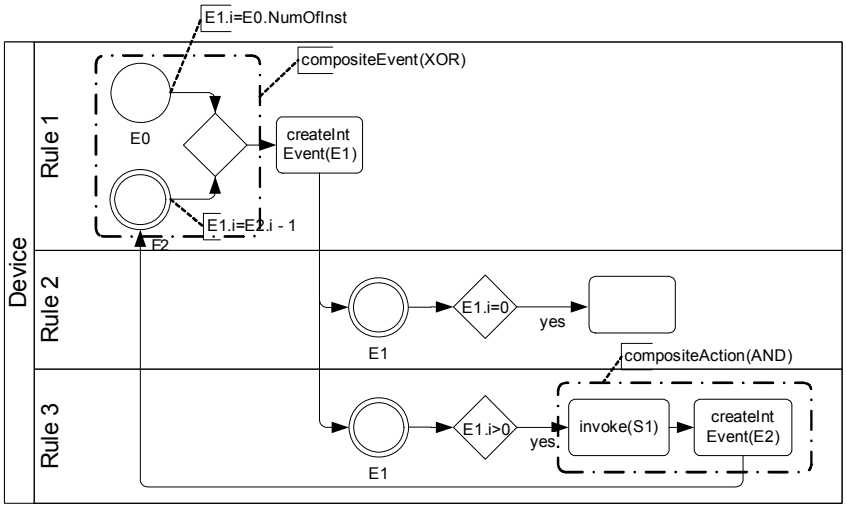
비동기적 다중 인스턴스(패턴 12)는 동기화되지 않는 다중 인스턴스 패턴으로 <그림 13>과 같이 표현할 수 있다. Rule 1의 시작시에 생성할 다중 인스턴스의 개수(NumOf-Inst)가 지정되면, Rule 3는 NumOfInst > 0가 만족하는 동안 S1를 호출함과 동시에 Rule 1을 반복 시작시키고 NumOfInst를 1만큼 감소시킨다. 그러나 NumOfInst = 0가 되면, 더 이상 Rule 3가 시작되지 않고 Rule 2가 시작되어 다른 후속 프로세스를 진행시킨다.

설계시 결정 다중 인스턴스(패턴 13)는 설계시에 지정된 횟수만큼 인스턴스를 반복시키는 패턴으로 Rule 1에 전달되는 입력값에 따라서 패턴 12와 마찬가지로 Rule 1에서 인스턴스를 반복시킬 수 있다. 그러나 패턴 12와는 달리 동기화 패턴이기 때문에 Rule 3의 복합 액션에서 S1이 종료되어야만 E2를 통해 Rule 1을 시작시킨다.

<그림 14>는 설계시 결정 다중 인스턴스(패턴 13)와 실행시 결정 다중 인스턴스(패턴 14)를 동시에 표현하고 있으며, Rule 1의 입력값이 설계시에 지정되었다면 패턴 13이며, 실시간에 지정되었다면 패턴 14이다. 규칙 기반 설계에서 서로 분리된 별도의 규칙들이 실행되기 때문에 외부에서 입력되는 값(NumOfInst)은 설계시든 실시간이든 동일하게 취급되기 때문에 패턴 13과 패턴 14의 구분이 무의미하다.

4.5 상태 기반 패턴

다섯 번째 카테고리는 상태 기반 패턴(State-based patterns)으로 사용자 결정 분기(Deferred Choice : 패턴 16), 병렬 경로 중재(Interleaved Parallel Routing : 패턴 17), 마일스톤(Milestone : 패턴 18)을 포함한다.



〈그림 14〉 설계시/실행시 결정 다중 인스턴스(패턴 13, 패턴 14)

사용자 결정 분기(패턴 16)는 사용자에게 분기의 선택을 맡기는 패턴이다. 규칙 기반 프로세스 표현에서 외부로부터 메시지 이벤트를 수령할 수 있으며, <그림 15>와 같이 입력된 메시지의 값에 따라서 Rule 2와 Rule 3가 선택적으로 실행되도록 설계함으로써 이 패턴을 설계할 수 있다.

병렬 경로 중재(패턴 17)는 분기와 병합이 동기화를 수행함으로써 병렬 분기가 순차적으로 실행되는 패턴이다. WS-ECA 규칙 기반 프로세스에서 분기와 병합의 동기화를 표현할 수 있는 수단이 없으므로 표현이 불가능하다.

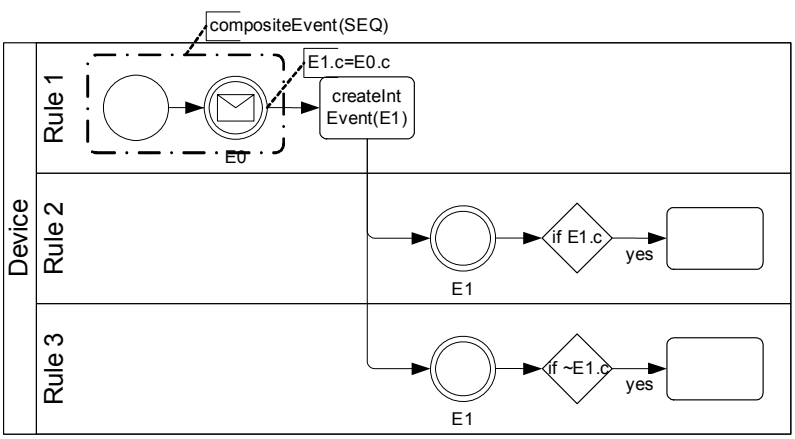
마일스톤(패턴 18)도 두 하위 프로세스 내의 액티비티 간의 동기화를 표현하는 수단이 제공되지 않기 때문에 WS-ECA 규칙으로 표현이 불가능하다.

4.6 취소 패턴

여섯 번째 카테고리는 취소 패턴(Cancellation patterns)으로 액티비티 취소(Cancel Activity : 패턴 19), 프로세스 취소(Cancel Case : 패턴 20)를 포함한다. ECA 규칙 기반 프로세스는 일반적인 비즈니스 프로세스와는 달리 상태를 가정하지 않기 때문에 취소 패턴이 적용될 수 없다. 즉, 이벤트나 서비스 결과에 대한 즉각적인 반응을 처리하는 능동적 규칙을 가정하고 있기 때문에, 취소를 하려면 실행 서비스 자체를 취소하기 위한 규칙을 만들어야 하지만, 이 또한 서비스가 취소 서비스를 제공하여야 한다. 이는 규칙 기반 설계의 범위를 벗어나므로 WS-ECA 규칙으로는 표현이 불가능하다고 할 수 있다.

5. 규칙 기반 프로세스의 WS-ECA 표현

본 절에서는 제 4장에서 제시된 ECA 규칙의 BPMN 표기를 WS-ECA 문서로 기술하는 방법을 설명한다. BPMN 표기에서 각 Pool은 WS-ECA 문서 하나에 대응되며, 제 3장에서 언급한 바와 같이 Pool과 Lane은



〈그림 15〉 사용자 결정 분기(패턴 16)

각각 WS-ECA 문서와 그 속에 포함된 <rule> 엘리먼트로 대응된다.

제4장에서 제시된 BPMN로 표기된 ECA 규칙들은 WS-ECA 언어로 직관적으로 변환이 가능하다. 예를 들어 단순 병합 패턴(패턴5)의 WS-ECA 표현 결과는 <그림 9>와 같이 표기된다. (a)의 내부 이벤트를 이용한 시스템 내부의 병합 구조를 표현하는 유형과, (b)의 외부 이벤트를 이용한 시스템 간의 병합 구조를 표현하는 유형은 각각 <그림 16>과 <그림 17>에 제시된 WS-ECA 문서로 변환될 수 있다.

앞서 설명한 WS-ECA 규칙 언어를 이용한 규칙 기반 프로세스 설계의 워크플로우 패턴 지원 유무는 <표 1>과 같이 정리할 수 있다. <표 1>은 비즈니스 프로세스를 설계하는 다른 언어들의 워크플로우 패턴 지원 유무와 비교하여 보여주고 있다[3, 14, 15, 17].

유비쿼터스 환경에서 여섯 가지 분류의 워크플로우 패턴들이 모두 의미가 있는 것은

```
<ECARule name = "Pattern_5_a"
    targetNampespace = "http://example.com/rules/exam_pt5_a"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    xmlns:pt = "http://example.com/patterns"
    xmlns = "http://di.snu.ac.kr/2005/eca/">
  <events>
    <intEvent name = "evt_a"/>
    <intEvent name = "evt_b"/>
    <compositeEvent type = "XOR" name = "evt_comp" TTL = "PT1H">
      <intEvent name = "E1"/>
      <intEvent name = "E2"/>
    </compositeEvent>
  </events>
  <actions>
    <createIntEvent name = "act_evt1" intEvent = "E1"/>
    <createIntEvent name = "act_evt2" intEvent = "E2"/>
    <invoke name = "act_invoke" service = "pt:S1"/>
  </actions>
  <rules>
    <rule name = "rule_1">
      <event name = "evt_a"/>
      <action name = "act_evt1"/>
    </rule>
    <rule name = "rule_2">
      <event name = "evt_b"/>
      <action name = "act_evt2"/>
    </rule>
    <rule name = "rule_3">
      <event name = "evt_comp"/>
      <action name = "act_invoke"/>
    </rule>
  </rules>
</ECARule>
```

〈그림 16〉 내부 이벤트를 이용한 단순 병합 패턴〈그림 9(a)〉의 WS-ECA 변환

```
<ECARule name = "Pattern_5_b1" ···>
  <events>
    <intEvent name = "evt_a"/>
    <extEvent name = "E1" eventID = "pt:E1"/>
  </events>
  <actions>
    <createExtEvent name = "act_evt" extEvent = "E1"/>
  </actions>
  <rules>
    <rule name = "rule_1">
      <event name = "evt_a"/>
      <action name = "act_evt"/>
    </rule>
  </rules>
</ECARule>
```

```
<ECARule name = "Pattern_5_b2" ···>
  <events>
    <intEvent name = "evt_b"/>
    <extEvent name = "E2" eventID = "pt:E2"/>
  </events>
  <actions>
    <createExtEvent name = "act_evt" extEvent = "E2"/>
  </actions>
  <rules>
    <rule name = "rule_2">
      <event name = "evt_b"/>
      <action name = "act_evt"/>
    </rule>
  </rules>
</ECARule>
```

```
<ECARule name = "Pattern_5_b3" ···>
  <events>
    <compositeEvent type = "XOR" name = "evt_comp" TTL = "PT1H">
      <extEvent name = "E1" eventID = "pt:E1"/>
      <extEvent name = "E2" eventID = "pt:E2"/>
    </compositeEvent>
  </events>
  <actions>
    <invoke name = "act_invoke" service = "pt:S1"/>
  </actions>
  <rules>
    <rule name = "rule_3">
      <event name = "evt_comp"/>
      <action name = "act_invoke"/>
    </rule>
  </rules>
</ECARule>
```

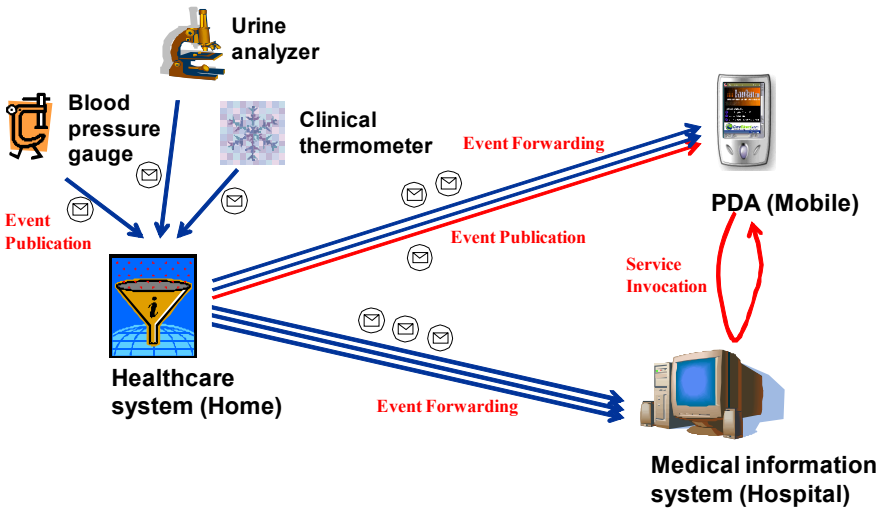
〈그림 17〉 외부 이벤트를 이용한 단순 병합〈그림 9(b)〉의 WS-ECA 변환

아니다. 그 중 다중 인스턴스 패턴들과 취소 패턴들은 유비쿼터스 환경에서 의미가 크지 않다. 먼저, 다중 인스턴스의 경우 즉각적으로 실행되는 유비쿼터스 서비스에서 거의 의미가 없다. 왜냐하면, 동일한 시스템의 동일한 서비스를 동시에 반복 요청한다는 것은 의미가 없기 때문이다. 예를 들어, 에어컨을 켜라는 명령을 동일한 에어컨에게 동시에 수차례 반복한다던가, 전등을 소멸하라는 요청을 동시에 수차례 반복한다는 것은 의미가

〈표 1〉 규칙 기반 프로세스 설계의 워크플로우 패턴 지원 여부

분 류	패 턴	WS-ECA	BPMN	BPEL	WSCI	UML2.0
기본적 통제 패턴	1. 직렬화	**Yes**	Yes	Yes	Yes	Yes
	2. 병렬적 분기	**Yes**	Yes	Yes	Yes	Yes
	3. 동기화	**Yes**	Yes	Yes	Yes	Yes
	4. 배타적 분기	**Yes**	Yes	Yes	Yes	Yes
	5. 단순 병합	**Yes**	Yes	Yes	Yes	Yes
고급 분기 및 동기화 패턴	6. 다중 분기	**Yes**	Yes	Yes	No	Yes
	7. 동기적 병합	**Yes**	Yes/No(1)	Yes	No	No
	8. 다중 병합	**Yes**	Yes	No	Yes/No	Yes
	9. 식별적 병합	**No**	Yes	No	No	Yes/No
구조적 패턴	10. 임의적 순환	**Yes**	Yes	No	No	Yes
	11. 암시적 종료	**No(2)**	Yes	Yes	Yes	Yes
다중 인스턴스 패턴	12. 비동기적 다중 인스턴스	**Yes(3)**	Yes	Yes	Yes	Yes
	13. 설계시 결정 다중 인스턴스	**Yes**	Yes	No	Yes	Yes
	14. 실행시 결정 다중 인스턴스	**Yes**	Yes	No	No	Yes
	15. 비결정적 다중 인스턴스	**No**	No	No	No	No
상태 기반 패턴	16. 사용자 결정 분기	**Yes(4)**	Yes	Yes	Yes	Yes
	17. 병렬 경로 중재	**No**	Yes/No	Yes/No	No	No
	18. 마일스톤	**No**	No	No	No	No
취소 패턴	19. 액티비티 취소	**No**	Yes	Yes	Yes	Yes
	20. 프로세스 취소	**No**	Yes	Yes	Yes	Yes

주) 1) Yes/No 표시는 간접적인 방법으로 표현이 가능하다고 볼 수도 있는 경우이다.
2) 표현이 불가능할 뿐만 아니라, 유비쿼터스 서비스에서는 실시간에 사용자 개입을 가정하고 있지 않는다.
3) 표현이 가능하지만, 동일한 서비스를 동시에 여러 번 호출하는 것은 유비쿼터스 컴퓨팅 환경에서 적합하지 않다.
4) 표현이 가능하지만, 유비쿼터스 서비스에서는 실시간에 사용자 개입을 가정하고 있지 않는다.



〈그림 18〉 유비쿼터스 헬스케어 시나리오

없다. 즉, 동일한 서비스 요청은 다른 시점이나 또는 다른 시스템에게 요청될 때 의미가 있다.

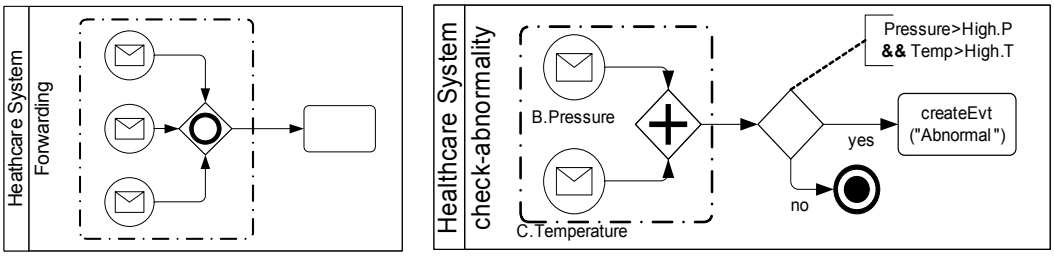
또한, 취소 패턴들도 유비쿼터스 환경에서는 의미가 없다. WS-ECA는 유비쿼터스 디바이스에서 즉각적으로 실행되는 것을 가정하여 무상태(stateless)로 실행되기 때문에, 취소시킬 인스턴스(instance)가 없다. 만약 이미 요청된 서비스에 대한 취소를 원한다면, 취소 서비스를 제공하는 시스템에 취소 이벤트를 처리하는 별도의 규칙을 만드는 것이 바람직하다.

6. 유비쿼터스 헬스케어의 규칙 기반 프로세스 설계

본 절에서는 유비쿼터스 서비스를 설계하

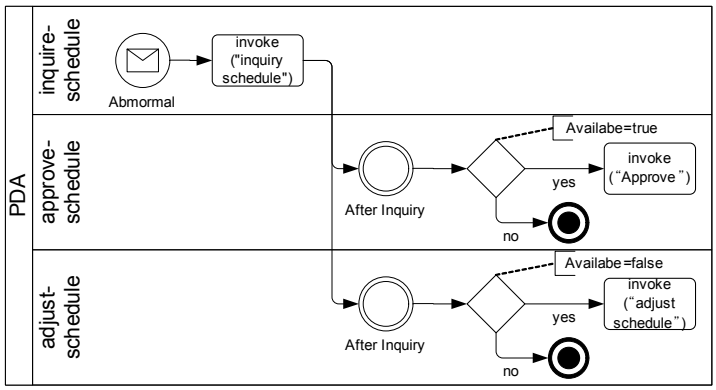
〈표 2〉 유비쿼터스 헬스케어 예제의 ECA 규칙

Rule Name	ECA Rule	Device
[Rule 1] Event forwarding	**On** B.Pressure ∨ C.Temperature ∨ U.Protein **Do** **createExtEvt**(“Healthcare data”)	Healthcare system
[Rule 2] Abnormality checking	**On** B.Pressure ∧ C.Temperature **If** B.Pressure > 150 **&&** C.Temp.>39 **Do** **createExtEvt**(“Abnormal health condition”)	Healthcare system
[Rule 3] Schedule checking	**On** **extEvt**(“Abnormal health condition”) **Do** invoke(“Inquiry available consultation hours”)	PDA
[Rule 4] Reservation	**On** **intEvt**(“Consultation reservation”) **Do** invoke(“Inquiry reservation”)	PDA



(a) [Rule1] Event forwarding (패턴 5)

(b) [Rule2] Abnormality checking(패턴 3)



(c) [Rule3] Checking schedule(패턴 2)

〈그림 19〉 유비쿼터스 헬스케어 예제 규칙의 BPMN 표기

기 위한 규칙 기반 프로세스의 시나리오를 제시하고, 프로세스 패턴의 유형을 추출한 후, BPMN으로 표기하고 WS-ECA 언어로 기술하는 예를 보여준다. WS-ECA를 이용한 규칙 기반 프로세스 설계를 적용하기 위한 예제로 유비쿼터스 헬스케어를 사용하였다[7]. <그림 18>는 유비쿼터스 환경에서 작동하는 측정기기 및 통신 장비들이 ECA 규칙에 따라 서비스를 진행하는 시나리오를 보여준다.

<그림 18>에 제시된 시나리오는 다음과 같다. 먼저, 혈압계, 체온계, 혈당검사기 등을 통하여 측정된 사용자의 건강 데이터는 헬스케어 시스템으로 전송되는데, 만약 이상 징후가 발견되면 해당병원의 의료정보 시스템과 사용자의 PDA에 이상 데이터를 전송하고, 사용자의 PDA는 의료정보 시스템을 통하여 방문일시를 예약하는 과정을 도와준다.

이 시나리오에 포함된 ECA 규칙을 구체적으로 나열하면 <표 2>와 같다. [Rule 1]은 혈압계(B), 체온계(C), 혈당측정기(U)에서 전달되는 혈압, 체온, 단백질 데이터를 매번 외부 이벤트로 생성시키는 규칙이다. 사용자는 필요에 따라 PDA, 핸드폰, 병원 등을 등록시켜 자동으로 데이터를 전송받을 수 있다. [Rule 2]는 혈압이 150mmHg 이상이면서 동

```
<ECARule name = "check-abnormality-rule"  targetNampespace = "http://example.com/healthcare-sys/rules/"
   xmlns:xsd = "http://www.w3.org/2001/XMLSchema"  xmlns = "http://di.snu.ac.kr/2005/eca/">
   <events>
      <extEvent name = "blood-pressure" eventID = "http://example.com/healthcare-sys/event/brood-presure"/>
      <extEvent name = "temperature" eventID = "http://example.com/healthcare-sys/event/temperature"/>
      <compositeEvent type = "AND" name = "pressure-and-temperature" TTL = "PT1H">
         <event name = "blood-pressure"/>
         <event name = "temperature"/>
      </compositeEvent>
      <extEvent name = "abnormal" eventID = "http://example.com/healthcare-sys/event/abnormal"/>
   </events>
   <actions>
      <createExtEvent name = "act_abnorm" extEvent = "abnormal"/>
   </actions>
   <rules>
      <rule name = "abnormal-condition-rule">
         <event name = "pressure-and-temperature"/>
         <condition expression = "/pressure/value &gt; 160 && /temperature/value &gt; 39"/>
         <action name = "abnormal-condition"/>
      </rule>
   </rules>
</ECARule>
```

〈그림 20〉 유비쿼터스 헬스케어 예제에서 Check abnormality 규칙의 WS-ECA 기술

시에 체온이 39°C 이상이면, 헬스케어 시스템이 "비정상 건강 상태"라는 외부 이벤트를 생성시키는 규칙이다. 본 시나리오에서는 의료정보시스템과 사용자 PDA에 이 외부 이벤트를 등록하고 있다고 가정하였다. [Rule 3]은 PDA가 "비정상 건강 상태"라는 메시지를 수령하면, 병원의 의료정보시스템에게 진료 가능한 시간을 요청하는 규칙이다. [Rule 4]는 사용자가 PDA를 통하여 진료 시간을 선택할 때 발생하는 내부 이벤트("Consultation reservation")가 발생하면, 의료정보시스템에게 예약을 요청하는 서비스를 요청하는 규칙이다.

위 네 가지 ECA 규칙은 각각 단순 병합(패턴 5), 동기화(패턴 3), 병렬적 분기(패턴 2)의 패턴을 포함하고 있는데, <그림 19>과 같이 BPMN 표기로 표현할 수 있다. 이 BPMN 표기들은 제 4장에서 언급한 바와 같이 WS-ECA 언어로 쉽게 변환이 가능한데, 그 중 [Rule 2] Abnormality checking 규칙을 WS-ECA 문서로 작성하면 <그림 20>과 같이 기술된다.

7. 결 론

본 연구에서는 유비쿼터스 환경에서 규칙

기반 프로세스를 설계하기 위한 방법을 워크플로우 패턴에 근거하여 분석하였다. 특히, 분석 대상은 이벤트-조건-액션이라는 ECA 규칙을 이용한 프로세스 설계이며, 설계 언어는 웹 서비스 기술을 활용하는 WS-ECA 언어를 사용하였으며, WS-ECA 언어 표현의 용이성을 위하여 프로세스 표준 표기법인 BPMN을 통하여 표현하였다.

프로세스 표현 방법을 분석하기 위한 워크플로우 패턴은 여섯 가지 분류로 20가지 정도가 있으며, ECA 규칙언어를 이용한 프로세스 설계 방법의 설계 가능 유무를 판정하고 설계 방법을 제시하였으며, 패턴들을 WS-ECA로 변환하는 예를 보여주었다.

프로세스 모델링은 Petri-net, EPC, UML Activity Diagram 등 다양한 모델과 언어를 사용하고 있다. 본 논문에서는 그 중에서 유비쿼터스 서비스 환경의 서비스 코디네이션을 위해서 적합한 ECA 규칙 기반의 프로세스 모델링 방법을 제시하였으며, 각 프로세스 패턴에 대하여 가능한 모델들을 제공함으로써 서비스 코디네이션을 실현하는 데 가이드라인을 제시하였다는 데 본 연구의 의의가 있다.

본 연구에서 분석 대상으로 하는 규칙 기반 프로세스 설계는 유비쿼터스 서비스 통합뿐만 아니라, 즉각적인 이벤트 처리가 요구되는 실시간 기업 환경에서도 효과적일 것으로 예상되므로, 규칙 기반 모델링의 프로세스 표현 가능성에 관한 분석 및 모델링 방법에 관한 본 연구가 유용하게 참고될 수 있을 것으로 기대된다.

참 고 문 헌

[1] W. M. P. van der Aalst, “The Application of Petri Nets to Workflow Management,” Journal of Circuits, Systems, and Computers, Vol. 8, No. 1, 1998, pp. 21-66.

[2] W. M. P. van der Aalst, “Formalization and Verification of Event-driven Process Chains,” Information and Software Technology, Vol. 41, No. 10, 1999, pp. 639-650.

[3] W. M. P. van der Aalst, A. H. M. ter Hofstede, B, Kiepuszewski, A.P. Barros, “Workflow Patterns,” Distributed and Parallel Databases, Vol. 14, No. 3, 2003, pp. 5-51.

[4] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel, “The WSML Rule Languages for the Semantic Web,” W3C Rules Workshop, 2005.http://www.w3.org/2004/12/rules-ws/paper/128/.

[5] M. Dumas and A. H. M. ter Hofstede, “UML Activity Diagrams as a Workflow Specification Language,” Proceeding of the 4th Conference on Unified Modeling Language(UML), 2001.

[6] J.-Y. Jung, J. Park, S.-K. Han, and K. Lee, “An ECA-based Framework for Decentralized Coordination of Ubiquitous Web Services,” Information and Software Technology, Vol. 49, No. 11-12, 2007, pp. 1141-1161.

[7] J.-Y. Jung, J. Park, S.-K. Han, and K. Lee, "Event-based Peer-to-Peer Process Enactment for Ubiquitous Web Service Devices," BPM Workshop on Grid and Peer-to-Peer Based Workflows, LNCS, Vol. 4103, 2006, pp. 387-399.

[8] K. Lee, W. Lee, J. Jeon, S. Lee, and J. Park, "Event-driven Coordination Rule of Web Services Enabled Devices in Ubiquitous Environments," W3C Workshop on the Ubiquitous Web, 2006.

[9] J. Mendling, G. Neumann, and M. Nuttgens, "Towards Workflow Pattern Support of Event-Driven Process Chains (EPC)," GI Workshop on XML for Business Process Management, 2005.

[10] OMG, Business Process Modeling Notation, V1.1, OMG Available Specification, OMG Document Number : formal/2008-01-17, 2008. http://www.omg.org/spec/BPMN/1.1/PDF.

[11] Rule Markup Language (RuleML), Rule Markup Initiative, http://www.ruleml.org/.

[12] J. de Bruijn, et al., Web Rule Language (WRL), W3C Member Submission, September 2005. http://www.w3.org/Submission/WRL/.

[13] S. White, "Process Modeling Notations and Workflow Patterns," in L. Fischer, ed., Workflow Handbook 2004, Future Strategies Inc., Lighthouse Point, FL, USA., 2004, pp. 265-294.

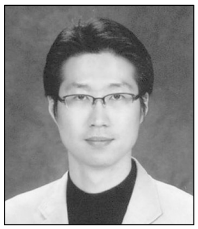
[14] P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "Pattern-based Analysis of BPEL4WS," QUT Technical Report, FIT-TR-2002-04, Queensland University of Technology, 2002.

[15] P. Wohed, B. Andersson, A. H. M. ter Hofstede, N. C. Russell, and W. M. P. van der Aalst, "Patterns-based Evaluation of Open Source BPM Systems : The Cases of jBPM, OpenWFE, and Enhydra Shark," BPM Center Report BPM-07-12, BPMcenter.org, 2007.

[16] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, "Pattern-based Analysis of the Control-Flow Perspective of UML Activity Diagrams," 24th International Conference on Conceptual Modeling (ER 2005), LNCS, Vol. 3716, pp. 63-78.

[17] Workflow Patterns Homepage. http://www.workflowpatterns.com/.

저 자 소 개



정재윤	(E-mail : jyjung@khu.ac.kr)
1999년	서울대학교 산업공학과 (학사)
2001년	서울대학교 산업공학과 (석사)
2005년	서울대학교 산업공학과 (박사)
2005년~2006년	네덜란드 아인트호벤공대 초빙연구원
2006년~2007년	서울대학교 유비쿼터스 컴퓨팅 원천기술 개발지원센터
2007년~현재	경희대학교 산업경영공학과 전임강사
관심분야	비즈니스 프로세스 관리(BPM), 유비쿼터스 서비스 컴퓨팅, 인터넷 비즈니스

박종헌	(E-mail : jonghun@snu.ac.kr)
1990년	서울대학교 산업공학과 (학사)
1992년	서울대학교 산업공학과 (석사)
2000년	Georgia Institute of Technology 산업시스템공학과 (박사)
1990년~1994년	대우자동차 주임연구원
1994년~1997년	제어계측신기술연구센터 선임연구원
2001년~2003년	Pennsylvania State University School of Information Sciences and Technology 조교수
2003년~2004년	KAIST 산업공학과 조교수
2004년~현재	서울대학교 산업공학과 교수
관심분야	인터넷 서비스, 엔터테인먼트 컴퓨팅, 모바일 서비스