ISSN 1738-7531

보안공학연구논문지

JSE

Journal of Security Engineering

Vol. 6, No. 6, December 2009

보안공학연구지원센터

보안공학연구논문지
Journal of Security Engineering

ISSN : 1738-7531 제6권,제6호,2009년12월

목 차

보안 응용

Web 2.0을 위한 임베디드 Open API 서비스 플랫폼 설계 및 구현 ····· 259
노영식, 변영철, 이동철

오픈 아키텍처 기반의 비즈니스 프로세스 프레임워크 ····· 409
서채연, 김영철

보안성 평가

정보보증제도 분석과 보안제품 적합성 평가체계의 설계 ····· 425
방영환, 고갑승, 신재인, 이강수

데이터 보호

안전한 OSGi 프레임워크에서의 실시간 데이터 스트림 처리 방법 ····· 205
차지윤, 변영철, 이동철

보안 설계

A 4-step approach to SCADA security ····· 463
Kum-Taek Seo, Farkhod Alisherov, Min-kyu Choi, Tai-hoon Kim

SCADA Network Insecurity: Securing Critical Infrastructures through SCADA Security Exploitation ···· 473
Giovanni A. Cagalaban, Yohwan So, Seoksoo Kim

보안공학연구논문지 논문투고안내

보안공학연구논문지 논문투고규정

논문 심사 규정

논문 발간 규정

포상 및 징계 규정

연구 윤리 규정

오픈 아키텍처 기반의 비즈니스 프로세스 프레임워크

서채연[1], 김영철[2]

Business Process Framework Based on The Open Architecture

Chae-Yun Seo[1], R. Young-Chul Kim[2]

요 약

기존 클로즈 아키텍처의 비즈니스 프로세스 모델링의 문제점을 개선하기 위해 오픈 아키텍처 기반의 비즈니스 프레세스 프레임워크를 제안한다. 제안한 오픈 아키텍처 기반의 비즈니스 프로세스 프레임워크는 각 레이어에 매핑된 하위레이어의 모든 정보접근을 통해, 빠른 업무수행이 가능하다.

그러기위해 BPM(Business Process Model)과 SOA(Service Oriented Architecture), 그리고 CBD(Component Based Development)를 접목하여 빠르게 변화하는 비즈니스 환경에 적응하고, 급변하는 비즈니스 프로세스 추가/개선을 쉽게 가능케 한다. 본 논문에서는 개선된 오픈 아키텍처의 5-Layer 구조와 오픈 아키텍처의 BNF(Backus Naur Form)을 정의하였다. 각 레이어의 접근에 대한 보안 문제도 고려 중 이다.

핵심어 : 컴포넌트, 서비스, 오픈 아키텍처, BPM, 프레임워크

Abstract

To improve the problem of BPM(Business Process Modeling) based on the closed architecture, we propose BPF(Business Process Framework) based on the open architecture. On our proposed BPF based on the open architecture, it will possibly execute quick business work through which the above layer will access all information of the subordinate layers mapped on it. This approach is to develop the right application with reusing software components in time and easily under rapid business process changing/improving through mapping BPM, SOA(Service Oriented Architecture), CBD(Component Based Development). We also define 5-Layers of BPF(business Process Framework), and an BNF(Backus Naur Form) of constructing BPF. Security problem will be considering to access on each layer.

Keywords : Component, Service, Open Architecture, BPM(Bussiness Process Modeling), Framework

1. 서론

빠르게 변화하는 IT기술은 최소한의 정보기술로 다양한 비즈니스 프로세스를 기업 내/외부의

접수일(2009년04월10일), 심사의뢰일(2009년04월11일, 심사완료일(1차:2009년05월12일, 2차:2009년 05월30일)

게재일(2009년12월31일)

[1]홍익대학교 일반대학원 소프트웨어공학전공.
email: jyun@selab.hongik.ac.kr

[2](교신저자) 홍익대학교 과학기술대학 컴퓨터정보통신 부교수.
email: bob@selab.hongik.ac.kr

요구에 쉽고 빠르게 전환하길 원한다. 그러나 기업의 대부분 시스템에는 복잡한 애플리케이션들로 혼재되어, 기존의 비즈니스 프로세스를 변경하거나 개발이 어렵다. 이러한 환경은 급변하는 시장에서 기업의 신속한 비즈니스 규칙 수립과 의사 결정을 지연시켜, 시장 경쟁력을 저해시키는 요인이 되고 있다[11]. 그러나, 민첩한 IT환경에서 SOA(Service Oriented Architecture)를 통해 서비스를 추출해서 독립적으로 구현하고, 구현된 서비스를 제공하여 새로운 비즈니스 프로세스를 창출이 가능해야한다[12]. 컴포넌트와 서비스의 재사용은 현대의 소프트웨어 개발의 많은 핵심 쟁점을 해결할 수 있는 잠재력이 있다. 이들의 재사용은 시장 변화에 대해 빠르게 반응하여 단기간 시장 진입이 가능하고, 개발과 관리 비용을 감소시킴으로써 향상된 품질을 만들 수 있다[2]. 컴포넌트와 서비스 재사용의 궁극적인 목적은 첫째는 향상된 생산성과 짧아진 리드타임을 통해 일을 줄일 수 있다. 둘째는 향상된 품질을 통해 에러의 수를 감소한다. 셋째는 향상된 생산성을 통해 유지보수 비용을 낮춘다. 넷째는 재사용 시 필요한 전체적인 요구분석을 통한 재작업을 줄이기 위한 것이다[13]. 컴포넌트 기반 개발이란 컴포넌트의 재사용으로 소프트웨어를 개발하는 방법론이다[11]. 개발된 컴포넌트는 컴포넌트 레파지토리에 저장하고, 저장된 컴포넌트는 필요한 사용자에 의해 재배치/재사용 가능하다[1].

본 논문은 클로즈 아키텍처의 비즈니스 프로세스를 오픈 아키텍처의 5-Layer 구조[3,6]로 개선을 제안한다. 기존 프레임워크는 클로즈 기반으로 한 레이어를 관리하기 위해 매핑된 모든 레이에 대한 정보를 검색해야 되는 반면 개선된 오픈 아키텍처는 한 레이어가 매핑된 모든 정보를 포함하기 때문에 한번에 검색이 가능하다. 2장 개선된 비즈니스 프로세스 프레임워크, 3장 오픈 레이어 구조를 위한 BNF를 설명하고, 4장에서는 결론 및 향후 과제에 대해 기술한다.

2. 비즈니스 프로세스 프레임워크

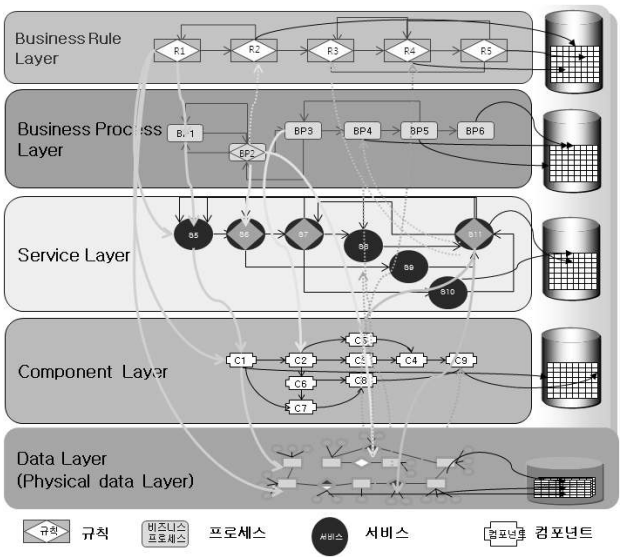
2.1 비즈니스 프로세스 프레임워크

기존의 비즈니스 프로세스 프레임워크는 클로즈 구조이다[2]. 이는 상하위 레이어가 연결되어 있다. 오픈 아키텍처는 하위의 모든 레이어에 대한 정보를 갖기 때문에 시스템 운영과 실행 시 효율성이 높다.

2.1.1 개선된 오픈 비즈니스 프로세스 프레임워크

개선된 비즈니스 프로세스 프레임워크는 오픈 아키텍처이다. 이 구조는 각 레이어와 데이터를 주고받을 수 있는 상호운용성과 호환성, 표준성, 유연성을 갖고 있어 더 빠른 데이터 처리가 가능하다. 그리고, 오픈 구조이기 때문에 사용자가 비즈니스 요구사항을 충족할 수 있는 맞춤형 전환이 쉽다[14-15]. 사용하기 쉽기 때문에 기존 시스템에 적용이 자유롭다. 각 레이어는 독립성을 유지하되 한부분의 변경이 다른 부분에 미치는 영향을 최소화 하여 안정적인 환경을 제공한다. 각 레이

어의 레파지토리는 테이블화하여, BPSQL(Business Process Structured Query Language)을 통해 각 레이어별 데이터 쿼리를 생성하여 필요한 정보를 추출하고자 한다. 레이어 구조는 비즈니스 요구 발생 시 기 존재하는 재사용 컴포넌트를 활용하여 신속하게 신규 서비스를 생성할 수 있으며, 생성된 서비스로 신규 비즈니스를 구성할 수 있다. [그림 1]은 비즈니스 프로세스 프레임워크이다. 오픈 아키텍처 기반으로 BPM과 서비스 그리고 CBD를 접목하여 5-Layer 구조로 개선한 것이다.



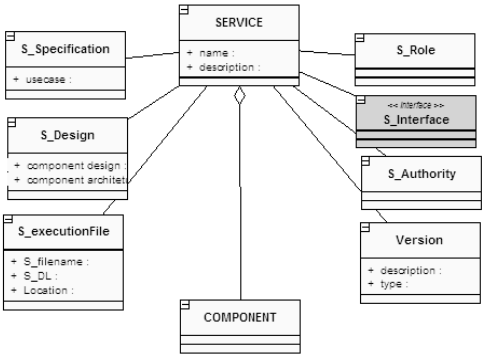
[그림 1] 오픈 아키텍처 기반의 비즈니스 프로세스 프레임워크

[Fig. 1] Business process framework based on open architecture

개선된 비즈니스 프로세스 레이어의 구조는 다음과 같다 : 최상위의 5-Layer는 비즈니스 규칙, 4-Layer는 비즈니스 프로세스, 3-Layer는 서비스 , 2-Layer는 컴포넌트, 마지막 레이어는 실질적인 데이터가 존재하는 데이터베이스 즉, 데이터 모델링 레이어 이다. 그리고 각 레이어 에는 레파지토리가 존재한다. 오픈 아키텍처는 자신의 하위 레이어의 대한 모든 정보를 포함하기 때문에 실행 시 효율성이 좋고, 각 레이어에 연관된 데이터를 한눈에 파악이 가능하다다. 각각의 레이어는 다음과 같이 정의한다.

2.1.2 비즈니스 규칙 레이어(Business Policy Layer)

비즈니스 규칙은 회사의 사업 방식을 구성하는 유효성 편집, 로그온 확인, 데이터베이스 찾아보기, 정책, 알고리즘적 변형 등의 조합이며, 비즈니스 논리라고도 한다[7]. 또는, 비즈니스의 어떤 모습을 정의하거나 제한사항을 나타내는 문장으로 비즈니스 정책과 로직을 기술한 것이다[8]. [그림 2]는 비즈니스 규칙을 클래스다이어그램의 UML로 표기하였다. 규칙은 각 프로세스와 그 하위 레이어에 영향을 주고, 그 규칙과 연관된 하위 레이어를 수행한다.



[그림 2] UML의 클래스다이어그램을 이용한 서비스 정의

[Fig. 2] Definition of service using the classdaigram of UML

2.1.3 비즈니스 프로세스 레이어 (Business Process Layer)

특정한 목표나 목적을 달성하기 위한 활동, 작업 및 절차들의 집합을 프로세스라고 한다[6].

Jacobson은 비즈니스 프로세스를 고객에게 봉사하기 위하여 실행되는 내부 활동의 셋이라 했고, Bider는 목표에 도달하기위해 부분적으로 요구된 활동의 집합이라 하였다[9]. 프로세스는 주어진 목적을 위해 수행되는 일련의 단계들로, 어떤 목적 달성을 위해 무엇을 어떻게 해야 하는지에 대한 구체적인 활동 절차들로 구성된다[10]. 개발자는 비즈니스 프로세스를 디자인하는데, 비즈니스의 요구사항이 변경에 쉽게 대처하기 위해 서비스들의 조합으로 쉽게 시스템을 구축할 수 있다. 또한, 변화에 유연한 비즈니스를 구축하여 비즈니스의 변화의 속도에 따른 적절한 프로세스를 구축할 수 있다. 비즈니스 프로세스는 스펙과 롤이 있으며, 하나의 서비스 또는 여러 개의 서비스로 구성된다. 각 프로세스는 고유의 이름이 있고, 스펙은 유스케이스로 표현가능하다.

2.1.4 서비스 레이어(Service Layer)

서비스 레이어는 서비스의 유연성, 가시성, 유용성을 갖는다[14]. 서비스의 유용성은 각 서비스 간 의존성을 최소화한 아키텍처이다. 가시성은 서비스의 흐름이 외부에 노출되어 이해가 쉽다. 유용성은 기 제작된 서비스 및 모듈의 재사용으로 개발자가 편리하고 쉽게 활용할 수 있다. 이러한 특징을 통해 급변하는 환경 변화에 능동적으로 대처할 수 있을 뿐 아니라 다양한 커스트 마이징 요건을 쉽게 수용할 수 있는 유연성을 보장한다[11]. 본 논문에서 서비스는 여러 개의 컴포넌트가 하나의 서비스를 만들고, 여러 서비스들은 하나의 비즈니스 프로세스를 만든다[12].

서비스는 스펙, 디자인, 실행 파일, 역할, 인터페이스, 인증, 버전을 갖는다. 이러한 서비스는 여러 개의 컴포넌트로 구성된다[2]. 서비스의 스펙은 유스케이스로 표현하고, 디자인에는 서비스를 구성하는 컴포넌트의 디자인과 구조를 포함한다. 실행 파일에는 파일 명과 묘사 언어가 있고, 로케이션을 통해 서비스를 구성하는 컴포넌트의 위치를 알 수 있다[6].

[표 1]은 서비스 레이어를 XML로 명세한 것이다.

[표 1] XML로 명세된 서비스
[Table 1] The service specified by XML

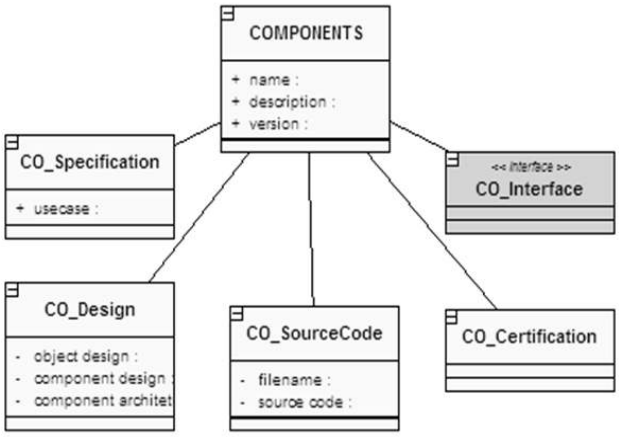
```
<?xmlversion="1.0" ncoding="UTF-8"?>
<!? Define Service Structure -->

<!DOCTYPE SERVICEINFORMATIONS
[
<!ELEMENT SERVICEINFORMATIONS (Service+) >
<!ELEMENT Service (name,description)
(Version+) >
<!ATTLIST Service
name                CDATA    #REQUIRED
description CDATA    #REQUIRED>
<!ELEMENT Policy
(PreCondition,PostCondition,Incariant)
>
<!ATTLIST Policy
PreCondition        CDATA    #REQUIRED
PostCondition       CDATA    #REQUIRED
Invariant           CDATA    #REQUIRED
<!ElEMENT Role>
<!ELEMENT Version
(verTag,config,description,langType)
(Specification+,DesignSpec+,SourceCode+,TestResult+,Certi
fication+,Wrapper*,OuterInterface+,InnerInterface+,)>
<!ATTLIST Version
verTag              CDATA    #REQUIRED
config              CDATA    #REQUIRED
description CDATA    #REQUIRED
langType    CDATA    #REQUIRED
>
<!ELEMENT Specification (usecase) >
<!ATTLIST Specification
usecase             CDATA    #REQUIRED
>
<!ELEMENT DesignSpec (object design, Service
design, Service architecture) >
<!ATTLIST DesignSpec
object design       CDATA    #REQUIRED
service design      CDATA    #REQUIRED
service architecture CDATA   #REQUIRED
>
<!ELEMENT SourceCode (fileName, sourceCode) >
<!ATTLIST SourceCode
fileName            CDATA    #REQUIRED
sourceCode CDATA     #REQUIRED
>
<!ELEMENT
TestResult(fileName,testResult) >
<!ATTLIST TestResult
fileName            CDATA    #REQUIRED
testResult  CDATA    #REQUIRED>
<!ELEMENT Certification () >
<!ELEMENT Adapter () >
<!ELEMENT Facade (name, role) >
<!ATTLIST Facade
    name CDATA    #REQUIRED
    role  CDATA    #REQUIRED
>
<!ELEMENT OuterInterface (name, role) >
<!ATTLIST OuterInterface
    name CDATA    #REQUIRED
    role  CDATA    #REQUIRED
>
<!ELEMENT InnerInterface (name, role) >
<!ATTLIST InnerInterface
    name CDATA    #REQUIRED
    role  CDATA    #REQUIRED>
]>
<SERVICEINFORMATIONS>
```

2.1.5 컴포넌트 레이어(Component Layer)

컴포넌트는 동적인 흐름을 갖는 워크플로우 형태이다. 컴포넌트는 여러 개의 컴포넌트, 컴포넌트 워크플로우로 구성된다. 컴포넌트는 내포된 하위컴포넌트를 가질 수 있다. 도메인 분석 방법론을 통해 추출한 컴포넌트(WODA)를 테이블화한 컴포넌트 레파지토리에 저장한다[4].

[그림 3]은 컴포넌트를 UML의 클래스 다이어그램으로 표현하였다. 컴포넌트는 스펙, 디자인, 소스코드, 인증, 인터페이스가 존재한다. 컴포넌트는 버전을 추가하면서 업데이트 할 수 있다. 컴포넌트 디자인에는 객체디자인과 컴포넌트디자인, 구조를 포함한다. 인증을 통해 유용한 컴포넌트를 식별 할 수 있다.



[그림 3] UML의 클래스다이어그램을 이용한 컴포넌트 정의
[Fig. 3] Definition of component using classdaigram of UML

2.1.6 데이터 레이어(Data Layer)

내부물리적인 공간에 데이터가 저장되는 곳이다. 상위 4-Layer가 수행되면서 실제 데이터를 사용한다. 데이터는 E-R 다이어그램으로 모델링되고, 레파지토리의 모든 데이터들은 테이블 화되어 저장한다. 이는 기존 질의 언어를 확장함으로써 접근성이 편리하고 빠른 정보 검색을 위해서이다. 하기 위함이다[10]. 데이터 레이어는 E-R 다이어그램으로 모델링한다.

3. 5-Layer구조를 위한 BNF (Backus Naur Form)

비즈니스 프로세스 프레임워크 내 각 레이어는 BNF를 이용하여 설계 한다.

오픈 구조는 실행 시 효율성을 높이기 위해 각 레이어에 대한 정보를 갖는다. 즉, 이벤트가 발생한 레이어와 이와 연관된 엑티비티들에 대한 정보도 포함한다. 각 레이어에 대한 정보를 표현하기 위하여 상위 레이어에 대한 정보를 SUPER-REFERENCE로 표기하였고, 하위 레이어는 SUB-REFERENCE로 표기하였다.

3.1 비즈니스 규칙 (Business Policy)

[그림 4]는 BPF(Business Process Framework)의 최상위층인 비즈니스 규칙 모델링 레이어의 BNF이다.

```
BPOLICY <policy-name>

{CATEGORY:
<RULE    |Rule-POLICY
 IN: {<RULE type>[: RULE PROCESS>],}


POLICY-SPECIFICATION:
    [preCOND: (condition-predicate},]
    [ACTION: {<action>,}]
    [postCOND: (condition-predicate},]}


LINK-REFERENCES:
     Policy:
       [CREATE: {<policy>,}]
       [USE: {< policy >,}]
       [UPDATE: {< policy >,}]
       [DELETE: {< policy >,}]
```

```
SUB-REFERENCES:
      BPROCESS:
        [CREATE: {<bprocess>,}]
        [USE: {< bprocess >,}]
        [UPDATE: {< bprocess >,}]
        [DELETE: {< bprocess >,}]
           SERVICE:
        [CREATE: {<SERVICE>,}]
        [USE: {<SERVICE>,}]
        [UPDATE: {<SERVICE>,}]
        [DELETE: {<SERVICE>,}]
            COMPONENT:
                [CREATE: {<component>, }]
        [USE: {<component>, }]
        [UPDATE: {<component>, }]
        [DELETE: {< component >, }]}
           DATA:
            [CREATE: {<data>, }]
            [USE: {<data>, }]
            [UPDATE: {<data>, }]
            [DELETE: {<data>, }] }
```

[그림 4] 비즈니스 규칙 레이어의 BNF

[Fig 4] BNF of the business rule layer

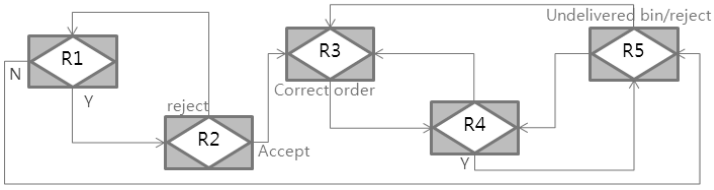
비즈니스 규칙에 맞게 비즈니스 프로세스 모델링에 적용한다. 룰 확인은 선 조건을 확인한 후, 액션을 취한다. 액션이 룰에 올바른지 검사하기 위해 후 조건을 체크한다. [표 2]는 비즈니스 규칙에 대한 표현기호이다.

[표 2] 비즈니스 규칙에 대한 기호

[Table 2] Sign for the business rule

표현기호	설 명
	규칙
	조건 규칙
	각 규칙을 연결하는 선

[그림 5]는 비즈니스 규칙을 BNF로 모델링하였다. R1은 등급, R2는 인증, R3은 확인, R4는 신용, R5는 정보에 관한 규칙들이다.



[그림 5] 비즈니스 규칙 모델링

[Fig. 5] Business rule modeling

3.2 비즈니스 프로세스 (Business Process)

[그림 6]은 비즈니스 프로세스 모델링 레이어의 BNF인 비즈니스 프로세스를 정의한 것이다.

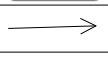
```
BPROCESS <bprocess-name>
{    [PART-OF: {<bprocess>,{]
     [CONTAINS: {
     [EVENT: EVENTS: {<event>,}]
      LINK -REFERENCES:
           BPROCESS:
           [CREATE: {<bprocess>,}]
           [USE: {< bprocess >,}]
           [UPDATE: {< bprocess >,}]
           [DELETE: {< bprocess >,}]
}
EVENT <event-name>
{    TYPE-OF:<INTERNAL|EXTERNAL>
     EVENT-SPECIFICATION:
     { COND: {<condition-predicate>,}
       ACTION:{<activity>,}
}
SUPER-REFERENCES:
Policy:
           [CREATE: {<policy>,}]
           [USE: {< policy >,}]
           [UPDATE: {< policy >,}]
           [DELETE: {< policy >,}]
LINK-REFERENCES:
BPROCESS:
           [CREATE: {<bprocess>,}]
           [USE: {< bprocess >,}]
           [UPDATE: {< bprocess >,}]
           [DELETE: {< bprocess >,}]
SUB-REFERENCES:
SERVICE:
           [CREATE: {<SERVICE>,}]
           [USE: {<SERVICE>,}]
           [UPDATE: {<SERVICE>,}]
           [DELETE: {<SERVICE>,}]
COMPONENT:
           [CREATE: {<component>, }]
           [USE: {<component>, }]
           [UPDATE: {<component>, }]
           [DELETE: {< component >, }]}
DATA:
           [CREATE: {<data>, }]
           [USE: {<data>, }]
           [UPDATE: {<data>, }]
           [DELETE: {<data>, }]}
}}
ACTIVITY <activity-name>
{ [PERFORM-BY: {<agent-names>,}]
  TRIGGERED-BY:{<event>,}
  PART-OF: {<process>,}
SUPER-REFERENCES:
Policy:
           [CREATE: {<policy>,}]
           [USE: {< policy >,}]
           [UPDATE: {< policy >,}]
           [DELETE: {< policy >,}]
LINK-REFERENCES:
BPROCESS:
           [CREATE: {<bprocess>,}]
           [USE: {< bprocess >,}]
           [UPDATE: {< bprocess >,}]
           [DELETE: {< bprocess >,}]
SUB-REFERENCES:
SERVICE:
           [CREATE: {<SERVICE>,}]
           [USE: {<SERVICE>,}]
           [UPDATE: {<SERVICE>,}]
           [DELETE: {<SERVICE>,}]
COMPONENT:
           [CREATE: {<component>, }]
           [USE: {<component>, }]
           [UPDATE: {<component>, }]
           [DELETE: {< component >, }]}
DATA:
           [CREATE: {<data>, }]
           [USE: {<data>, }]
           [UPDATE: {<data>, }]
           [DELETE: {<data>, }]}
```

[그림 6] 비즈니스 프로세스 레이어의 BNF

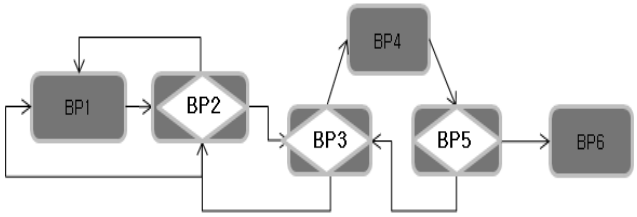
[Fig. 6] BNF of the business process layer

[표 3]비즈니스 프로세스 표현기호

[Table 3] Sign for the business process

표현기호	설 명
	비즈니스 프로세스
	분기가 있는 비즈니스 프로세스
	각 비즈니스 프로세스를 연결하는 선

[그림 7]은 물품을 구매하기 위한 비즈니스 프로세스이다.



[그림 7] 비즈니스 프로세스
[Fig. 7] Business process

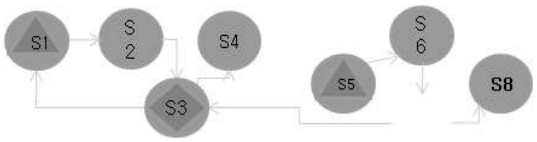
3.3 서비스 (Service)

비즈니스 프로세스 하위 레이어에서 실행되는 서비스들이다. 본 논문에서 정의한 서비스 스펙을 간단히 살펴보면, 여러 개의 컴포넌트로 구성된 서비스에는 이름과 설명이 있다. 서비스 버전을 통해 새로운 서비스와 구분 할 수 있고, 서비스 인터페이스를 통해 각 서비스와 커뮤니케이션 할 수 있다. [표 4]는 서비스 표현 기호에 대한 설명이다.

[표 4] 서비스 표현기호
[Table 4] Sign for service

표현기호	설 명
	서비스
	조건 서비스
	각 서비스를 연결하는 선

[그림 8]은 서비스 모델링에 대한 시나리오이다.



[그림 8] 서비스 모델링
[Fig. 8] Service modeling

[그림 9]는 BPF의 서비스층 BNF를 정의한다.

SERVICE<service-name>

{ [PART-OF: {<service>, {]
[CONTAINS: {
[EVENT: EVENTS: {<event>, }

LINK-REFERENCES:
SERVICE:
[CREATE: {<service>, }]
[USE: {<service>, }]
[UPDATE: {<service>, }]
[DELETE: {<service>, }]

EVENT <event-name>
{ TYPE-OF:<INTERNAL|EXTERNAL>
EVENT-SPECIFICATION:
{ COND: {<condition-predicate>, }
ACTION: {<activity>, } }

LINK-REFERENCES:
SERVICE:
[CREATE: {<service>, }]
[USE: {<service>, }]
[UPDATE: {<service>, }]
[DELETE: {<service>, }]
SUPER-REFERENCES:
Policy:
[CREATE: {<policy>,}]
[USE: {< policy >,}]
[UPDATE: {< policy >,}]
[DELETE: {< policy >,}]
BPROCESS:
[CREATE: {<bprocess>,}]
[USE: {< bprocess >,}]
[UPDATE: {< bprocess >,}]
[DELETE: {< bprocess >,}

SUB-REFERENCES:
COMPONENT:
[CREATE: {<component>, }]
[USE: {<component>, }]
[UPDATE: {<component>, }]
[DELETE: {< component >, }]}
DATA:
[CREATE: {<data>, }]
[USE: {<data>, }]
[UPDATE: {<data>, }]
[DELETE: {<data>, }]}
}}
ACTIVITY <activity-name>
{ [PERFORM-BY: {<agent-names>, }]
TRIGGERED-BY: {<event>, }
PART-OF: {<process>, }
SUPER-REFERENCES:
Policy:
[CREATE: {<policy>,}]
[USE: {< policy >,}]
[UPDATE: {< policy >,}]
[DELETE: {< policy >,}]
BPROCESS:
[CREATE: {<bprocess>,}]
[USE: {< bprocess >,}]
[UPDATE: {< bprocess >,}]
[DELETE: {< bprocess >,}]
SUB-REFERENCES:
COMPONENT:
[CREATE: {<component>, }]
[USE: {<component>, }]
[UPDATE: {<component>, }]
[DELETE: {< component >, }]}
DATA:
[CREATE: {<data>, }]
[USE: {<data>, }]
[UPDATE: {<data>, }]
[DELETE: {<data>, }]}

[그림 9] 서비스 레이어의 BNF

[Fig. 9] BNF of service layer

3.4 컴포넌트 (Component)

[표 5]는 모델링된 서비스를 설명하는 것으로서, 물품 구매 프로세스의 하위 레이어의 서비스들 중 모델링 된 서비스를 설명한다.

[표 5] 컴포넌트 표현기호

[Table 5] Sign for component

표현기호	설 명
	컴포넌트
	조건 컴포넌트
	각 컴포넌트를 연결하는 선

[그림 10]은 컴포넌트 모델링 레이어의 BNF를 정의한다. 컴포넌트는 WODA(Workflow Oriented Domain Analysis)로 추출되고[5], 정의한 컴포넌트 스펙을 이용하여 작성한다. 컴포넌트는 여러 개의 컴포넌트, 컴포넌트 워크플로우로 구성된다. 컴포넌트는 내포된 하위컴포넌트를 가질 수 있다.

```
COMPONENT<component-name>
{   [PART-OF: {<component>, {]
    [CONTAINS: {
    [EVENT: EVENTS: {<event>, }
    LINK-REFERENCES:
     COMPONENT:
            [CREATE: {<component>, }]
            [USE: {<component>, }]
            [UPDATE: {<component>, }]
            [DELETE: {<component>, }]
EVENT <event-name>
{   TYPE-OF:<INTERNAL|EXTERNAL>
    EVENT-SPECIFICATION:
       {  COND: {<condition-predicate>, }
          ACTION: {<activity>, } }
    LINK-REFERENCES:
       COMPONENT:
            [CREATE: {<component>, }]
            [USE: {<component>, }]
            [UPDATE: {<component>, }]
            [DELETE: {<component>, }]
SUPER-REFERENCES:
Policy:
       [CREATE: {<policy>,}]
       [USE: {< policy >,}]
       [UPDATE: {< policy >,}]
       [DELETE: {< policy >,}]
BPROCESS:
       [CREATE: {<bprocess>,}]
       [USE: {< bprocess >,}]
       [UPDATE: {< bprocess >,}]
       [DELETE: {< bprocess >,}]
SERVICE:
       [CREATE: {<SERVICE>,}]
       [USE: {<SERVICE>,}]
       [UPDATE: {<SERVICE>,}]
       [DELETE: {<SERVICE>,}]
SUB-REFERENCES:
DATA:
       [CREATE: {<data>, }]
       [USE: {<data>, }]
       [UPDATE: {<data>, }]
       [DELETE: {<data>, }]}
}}
ACTIVITY <activity-name>
{ [PERFORM-BY: {<agent-names>, }]
  TRIGGERED-BY: {<event>, }
  PART-OF: {<component>, }
  LINK-REFERENCES:
    COMPONENT:
       [CREATE: {<COMPONENT>,}]
       [USE: {<COMPONENT>,}]
       [UPDATE: {<COMPONENT>,}]
       [DELETE: {<COMPONENT>,}]
SUPER-REFERENCES:
Policy:
       [CREATE: {<policy>,}]
       [USE: {< policy >,}]
       [UPDATE: {< policy >,}]
       [DELETE: {< policy >,}]
BPROCESS:
       [CREATE: {<bprocess>,}]
       [USE: {< bprocess >,}]
       [UPDATE: {< bprocess >,}]
       [DELETE: {< bprocess >,}]
SERVICE:
       [CREATE: {<SERVICE>,}]
       [USE: {<SERVICE>,}]
       [UPDATE: {<SERVICE>,}]
       [DELETE: {<SERVICE>,}]
SUB-REFERENCES:
DATA:
       [CREATE: {<data>, }]
       [USE: {<data>, }]
       [UPDATE: {<data>, }]
       [DELETE: {<data>, }]}
}}
```

[그림 10] 컴포넌트 레이어의 BNF

[FIg. 10] BNF of component layer

3.5 데이터 (Data)

데이터 모델링은 시스템을 구축하는데 필요한 각각의 데이터들을 E-R 다이어그램으로 표현한다. 구매프로세스에 대한 시스템을 구축하기 위해선 회원 관리 테이블, 판매테이블, 재고테이블, 배송테이블, 결제테이블, 물품테이블 등으로 구성된다. [그림 11]은 BPF의 최하위층인 데이터 레이어의 BNF이다.

```
DATA <DATA-name>
{[PART-OF: {<data>, {]
 [CONTAINS: {
 [EVENT:
     START-EVENT: {<event>, }
     EVENTS: {<event>, }
     END-EVENT: {<event>, }]
LINK-REFERENCES:
     DATA:
     [CREATE: {<data>, }]
     [USE: {<data>, }]
     [UPDATE: {<data>, }]
     [DELETE: {<data>, }]
EVENT <event-name>
{
     EVENT-SPECIFICATION:
     { COND: {<condition-predicate>, }
       ACTION: {<activity>, } }
SUPER-REFERENCES:
Policy:
          [CREATE: {<policy>,}]
          [USE: {< policy >,}]
          [UPDATE: {< policy >,}]
          [DELETE: {< policy >,}]
BPROCESS:
          [CREATE: {<bprocess>,}]
          [USE: {< bprocess >,}]
          [UPDATE: {< bprocess >,}]
          [DELETE: {< bprocess >,}]
SERVICE:
          [CREATE: {<SERVICE>,}]
          [USE: {<SERVICE>,}]
          [UPDATE: {<SERVICE>,}]
          [DELETE: {<SERVICE>,}]
COMPONENT:
            [CREATE: {<component>, }]
            [USE: {<component>, }]
            [UPDATE: {<component>, }]
            [DELETE: {<component>, }]
SUB-REFERENCES:
DATA:
          [CREATE: {<data>, }]
          [USE: {<data>, }]
          [UPDATE: {<data>, }]
          [DELETE: {<data>, }]}
}}
```

```
LINK-REFERENCES:
     DATA:
     [CREATE: {<data>, }]
     [USE: {<data>, }]
     [UPDATE: {<data>, }]
     [DELETE: {<data>, }] }
ACTIVITY <activity-name>
{ [PERFORM-BY: {<agent-names>, }]
  START-TIME:<time>
  END-TIME:<time>
  TRIGGERED-BY: {<event>, }
  PART-OF: {<process>, }
LINK-REFERENCES:
   DATA:
     [CREATE: {<data>, }]
     [USE: {<data>, }]
     [UPDATE: {<data>, }]
     [DELETE: {<data>, }] }
SUPER-REFERENCES:
Policy:
          [CREATE: {<policy>,}]
          [USE: {< policy >,}]
          [UPDATE: {< policy >,}]
          [DELETE: {< policy >,}]
BPROCESS:
          [CREATE: {<bprocess>,}]
          [USE: {< bprocess >,}]
          [UPDATE: {< bprocess >,}]
          [DELETE: {< bprocess >,}]
SERVICE:
          [CREATE: {<SERVICE>,}]
          [USE: {<SERVICE>,}]
          [UPDATE: {<SERVICE>,}]
          [DELETE: {<SERVICE>,}]
COMPONENT:
            [CREATE: {<component>, }]
            [USE: {<component>, }]
            [UPDATE: {<component>, }]
            [DELETE: {<component>, }]
SUB-REFERENCES:
DATA:
          [CREATE: {<data>, }]
          [USE: {<data>, }]
          [UPDATE: {<data>, }]
          [DELETE: {<data>, }]}
```

[그림 11] 데이터 레이어의 BNF

[Fig. 11] BNF of data layer

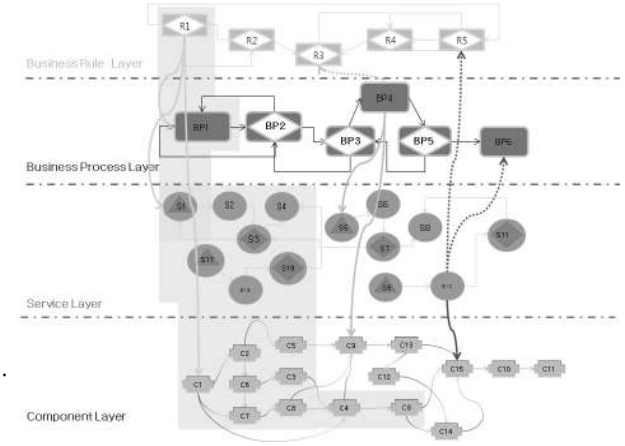
3.6 개선된 비즈니스 프로세스 프레임워크 내의 layer들 간의 접목

다음은 신용카드회사의 결제 프로세스에 대한 적용사례이다. BPF로 구조를 정의하여 결제 프로세스의 표준화와 향후 업무 개선에 효율적으로 대처하기 위한 기반을 마련하고자 한다.

[그림 12]는 결제에 대한 비즈니스 프로세서 프레임워크이다. 결제 시스템의 5-Layer는 룰과 프로세스, 서비스, 컴포넌트의 구조를 갖는다. 최상위는 비즈니스 프로세스 룰을 모델링한다. 각 룰을 수행하는 비즈니스 프로세스를 모델링하고, 각 비즈니스 프로세스에서 서비스를 추출하여 서비스

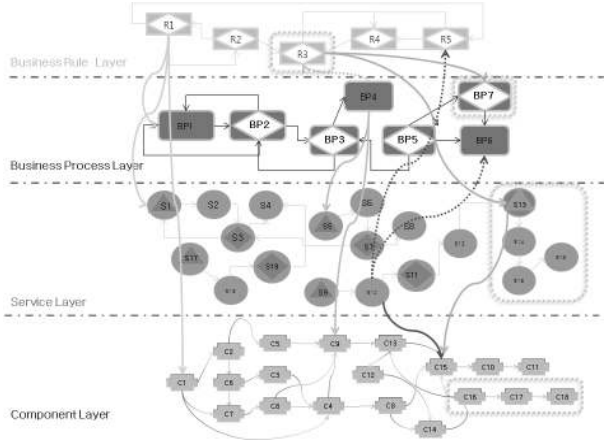
를 모델링한다. 서비스는 하나의 컴포넌트 또는 여러 컴포넌트로 생성할 수 있다. 바탕색이 칠해진 부분이 회원가입 규칙에 연관된 레이어를 표시한 것이다. 오픈 아키텍처는 연관된 레이어의 정보 파악이 용이하다. 결제 프로세스 중 결제 서비스는 오직 현금으로만 거래된다. 신용카드 결제를 생성하고, 각 레파지토리에 저장된 서비스와 컴포넌트를 추가하여 새로운 결제 프로세스를 개발한다.

[그림 13]은 기존 업무 프로세스의 변경으로 필요한 업무환경을 제공해주기 위해 소프트웨어 재사용을 통해 개발한 사례이다. 사용자의 필요에 의해 결제에 대한 비즈니스 규칙을 수정하였다. 새로운 비즈니스 규칙을 수행할 프로세스를 만들고, 모든 규칙에 매핑되는 서비스와 컴포넌트도 재배치하였다. 새로 생성된 것들은 점선의 사각형으로 표시하였다.



[그림 12] 결제에 대한 오픈 아키텍처의 5-Layer 구조

[Fig. 12] 5-Layer of open architecture for the payment by charge



[그림 13] 수정된 비즈니스 규칙에 맞게 서비스와 컴포넌트를 추가하여 만든 결제 프로세스

[Fig. 13] Payment process made by adding component and service fitted for the revised business rule

각 레이어의 점선 테두리로 표시된 것이 수정된 비즈니스 규칙과 새로 추가된 비즈니스 프로세스, 서비스, 컴포넌트이다. 시스템을 레이어 구조로 정의하여 설계하였기 때문에 편리한 동적설계가 가능하고, 모든 레이어에 대한 정보접근이 용이하다. 이는 비즈니스 환경 변화에 민첩하게 대응할 수 있는 시스템 구축 또한 가능하다. 프레임워크 구축으로 각 레이어간 상호 작용으로 원활한 관리 및 정보 검색 효율화를 이루고 각 프로세스 간 접근 용이성이 좋다. 오픈 프레임워크에 의해 업무 능률을 향상시킬 수 있다.

4. 결론

본 논문에서는 효율적인 비즈니스 프로세스를 위해 개선된 비즈니스 프로세스 프레임워크 5-Layer 구조를 제안하였다. 전체 시스템을 레이어 아키텍처로 구성함으로써 각 레이어의 동적/정적인 구조를 설계할 수 있고, 전체 시스템에서 연관된 모든 레이어의 파악이 가능하다. 시스템 구축 시 레이어 간 비즈니스 프로세스와 서비스의 접목을 통해 새로운 업무를 개발한다.

레파지토리에 저장된 컴포넌트와 서비스의 재사용으로 서비스와 비즈니스 프로세스를 구현 하면 개발 시간과 비용을 절약할 수 있다. 향후 비즈니스 프레임워크 내 데이터와 정보를 검색할 수 있는 BPSQL을 연구하고, 이를 사용한다면 모든 레이어가 연결 가능하다.

감사의 글

본 연구는 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업 (NIPA-2009-(C1090-0903-0004))과 교육과학기술부와 한국산업기술진흥원의 지역혁신인력양성사업으로 수행된 연구결과임.

참고문헌

[1] 서윤숙, 김영철, “확장된 BPM과 컴포넌트 기반방법론접목에 관한 연구”,홍익대학교, 2005.

[2] 서윤숙, 김영철, “A Closed Architecture기반의 3Layer",한국모바일학회, 2006.11.

[3] 서채연, 김동우, 김재수, 김영철,“효율적인 비즈니스 프로세스 모델링을 위한 5-Layer Architecture”, IWIT Vol. 6, No. 1, 19-22, 2008.5.

[4] 김윤정, 워크플로우 메커니즘을 통한 소프트웨어 컴포넌트 식별 방법론에 관한 연구, 홍익대학교, 2004.

[5] Howard Smith. "Business Process Management". 시그마인사이트컴. 2004.

[6] 서채연, 김동우, 김영철, “클로즈 아키텍처 기반의 비즈니스 프로세스 프레임워크”, 한국산학기술학회 Vol. 10, No. 8, 1939~1946, 2009.8.

[7] 노형진, 경영과학, 한올출판사, 2008.2

[8] 강상승, 손주찬, 함호상, 비즈니스 규칙 시스템과 객체 모델. KIPS 2003, 추계학술발표논문집, 2003 Nov. 14, pp.1205-1208.

[9] Jacobson,.I., the object advantage, addison-wesley,1995

[10] Bider, Business process modeling-concept, Proceedings of PBPM'00,2000

[11] Oracle SOA Architect Forum, 2009.1.8

[12] Bea Aqualogic BPM2007(why BPM + SOA), 2007.06.19.

[13] Hedley Apperly,Ralph Hofman etc, Service- and Component-Based Development: Using the Select Perspective and UML, Addison-Wesley, 2003.

[14] Tomas Earl, Service-oriented architecture : Concepts, technology, and design, 2006.

[15] Tomas Earl, Service-oriented architecture : A field guide to integrating XML and web service, 2006.

저자 소개

서채연 (Chae-Yun Seo)

2005년 2월 : 홍익대학교 일반대학원 소프트웨어공학 (공학석사)
2008년 2월 : 홍익대학교 일반대학원 소프트웨어공학 (박사수료)
2009년 3월 ~ 현재 : 유한대학 경영정보 겸임교수 및 강의 전담 강사
관심분야 : 컴포넌트, 서비스, 오픈 아키텍처, BPM, 프레임워크

김영철 (R. Young-Chul Kim)

2000년 : Illinois Institute of Technology(공학박사)
2000년 ~ 2001 : LG 산전 중앙연구소 Embedded system 부장
2001년 ~ 현재 : 홍익대학교 컴퓨터정보통신 교수
관심분야 : 테스트 성숙도 모델, 임베디드 S/W 개발 방법론 및 도구 개발, 상호 운영성 시험