웹 서비스 기반 비즈니스 프로세스 관리를 위한 Backward Recovery 방법

이순재*, 윤장혁**, 김광수***

A backward recovery method for web service driven business process management

Sunjae Lee*, Janghyeok Yoon**, Kwangsoo Kim***

Abstract

WS-BPMS(web service driven business process management system)은 기업 내 또는 비즈니스 파트너들 간의 비즈니스 프로세스를 체계적으로 통합, 관리하여 프로세스를 개선하고 자동화한다. 이는 비즈니스 프로세스를 검색한 후 BPEL4WS, BPML등의 프로세스 메타모델을 이용하여 새 프로세스를 디자인하고 BPMS위에 전개 실행하는 과정을 거치게 된다. 디자인된 프로세스는 복잡하여 그 실행에 있어 체계적이고 확고한 기반을 바탕으로 운영되어야 한다. 이를 위해 본 논문에서는 BPMS의 운영에 있어 pi-calculus를 기반으로 한 structured exception handling 방법론과 transaction 모니터링을 통한 backward recovery 방법을 제시한다.

***Key Word** : Business Process, Backward Recovery, BPEL4WS*

* 포항공과대학교 산업공학과 박사과정

** 포항공과대학교 산업공학과 석사과정

*** 포항공과대학교 산업공학과 정교수

1. Introduction

웹 서비스 기반 비즈니스 프로세스 관리 시스템은 기업 내 또는 비즈니스 파트너들 간의 비즈니스 프로세스를 체계적으로 통합, 관리하는 시스템을 말한다. 편의를 위해 본 논문에서는 WS-BPMS라고 하자. WS-BPMS의 목적은 비즈니스 프로세스를 컴포넌트화 하여 프로세스들의 재사용성을 높임으로써, 개별 업무가 아니라 프로세스 수준에서의 생산성을 향상, 개선시키고 동시에 이런 모든 과정을 자동화하는데 있다[2].

WS-BPMS하에서 비즈니스 프로세스 관리는 비즈니스 프로세스들의 검색, 설계, 전개, 실행 등의 기능들을 제공하여야 한다. WS-BPMS를 이용한 비즈니스 프로세스 관리의 가장 전형적인 시나리오는 1)비즈니스 프로세스들을 검색, 2)BPEL4WS, BPML등의 프로세스 메타모델을 이용하여 새 프로세스를 설계, 3)WS-BPMS위에 전개, 4)전개된 프로세스를 실행, 5)실행된 프로세스의 분석, 및 6) 분석결과를 반영한 프로세스 재설계하는 과정이지만, 반드시 이러한 순서대로 진행되어야 하는 것은 아니다[2]. 보다 효과적으로 관리하는 것은 순서에 구애 받지 않고 각 단계들을 적절한 시점에 수행하는 것이다[2].

설계 또는 재설계된 프로세스는 때때로 매우 복잡하기 때문에, 전개된 프로세스의 실행과 transaction 관리에 있어 WS-BPMS는 체계적이고 확고한 기반을 바탕으로 운영되지 않으면 안 된다[2]. 이러한 기반을 제공하기 위해 현재 WS-BPMS의 실행과 transaction 관리에 대한 많은 방법론들이 있으며, 과거의 접근법들 중 현재의 WS-BPMS와 가장 유사한 방법론들을 제공했던 것은 workflow management system(WFMS)이었다. 현재 workflow 및 WS-BPM 분야에서 주요 논점은 '어떻게 복잡한 프로세스의 올바르고 신뢰할 수 있는 실행을 보장할 것인가?' 하는 것이다. 이러한 문제에 대한 해결 방안으로 workflow에서는 forward execution, backward recovery, forward recovery 등의 접근법을 사용해 왔다[3]. 이들 중, 가장 기본이 되는 것이 바로 backward recovery이다. 본 논문에서는 WS-BPMS의 운영의 주요 근간인 pi-calculus 기반 structured exception handling을 기초로 하여[1,4], WS-BPM에서의 backward recovery 방법론을 제시하고자 한다.

2장에서는 관련 연구에 대해서 개관하고, 3장에서는 backward recovery를 위해 기존의 activity life cycle diagram(ALCD)을 다소 수정한 새 ALCD을 제시한다. 그리고 4장에서는 WS-BPM에서 제시한 pi-calculus기반의 structured exception handling architecture를 활용한 backward recovery architecture에 대해서 언급한다. 그리고 5장에는 토의와 결론, 6장에는 참조문헌이 이어진다.

2. 관련 연구

2.1 기본 정의들

2.1 Simple web services (activity)

Simple web services는 WSDL, UDDI, SOAP으로 구성된 1개의 단위 웹 서비스이다. 웹 서비스 제공자에 의해 제공되며 더 이상 분해할 수 없고 내부 처리 또한 밖으로 공개되지 않는다. 인터페이스만이 WSDL과 SOAP의 형태로 공개되어 웹 서비스가 다른 웹 서비스들을 이용할 수 있도록 하는 구조를 제공한다.

2.1.2 Complex web services (scope)

XLANG과 WSFL을 통합한 메타모델인 BPEL4WS에서는 scope을 1개 이상의 simple web services를 바탕으로 그 흐름을 구성한 단위로 정의한다[5]. 본 논문에서는 이를

complex web services라 하자.

2.2 Activity(scope) specification

Activity specification을 이루는 3가지 구성 요소는 다음과 같다[6].

(1) 외부에 공개된 activity의 execution states의 집합

(2) 이 states사이의 transitions의 집합

(3) Transition을 가능하게 하는 조건들

이 3가지 구성요소들은 ALCD로 표현된다[1]. BPEL4WS에서 제시한 Business-Agreement protocol state diagram이 가장 발전된 ALCD로, 이는 figure 1에 나타나 있다[5].

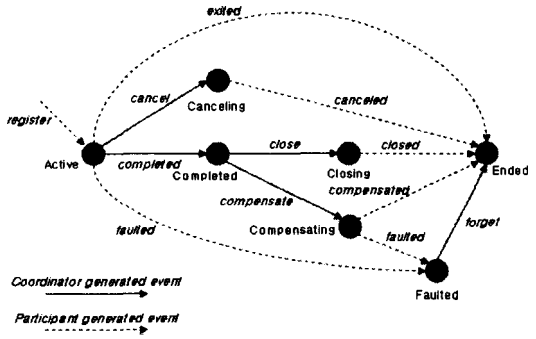

Figure 1. Activity life cycle diagram의 예

하지만, 이 ALCD은 2가지의 큰 문제점을 가지고 있다. 본 논문의 3장에서 이 문제점들을 언급하고 개선된 ALCD을 제시한다.

2.3 Backward recovery

Failure recovery 과정이란, complex web services내 simple web services에서 failure가 일어난 경우, 모든 complex와 simple web services가 최종적으로 ALCD에서 정의한 acceptable states에 도달하게 하는 것이다[6]. 원인이 되는 여러 failure들 중 semantic failure란 비즈니스 프로세스 내의 exception들로 이루어 져서 activity의 부정적 종료를 야기하는 것을 말한다[3]. 예를 들면 이미 모든 예약이 가득 차 있어 예약에 실패하거나, 사용자에 의해 예약 과정이 중단되는 경우이다[3]. Semantic failure에 대해 activity를 수행하기 이전 단계로 복구시키는 것을 backward recovery라 한다. 본 논문에서는 semantic failures에 대한 backward recovery 방법론을 다루게 된다.

3. Modified activity life cycle diagram

3.1 BPEL4WS ALCD의 문제점

앞서 언급한 바와 같이 BPEL4WS의 ALCD는 크게 2가지 문제점을 가진다. 첫 번째 문제점은 logging에 효율적이지 않다는 것이다. Activity는 의미적으로 볼 때, 주어진 일을 성공적으로 종료한 activity와 그렇지 못한 activity로 나뉠 수 있다[3]. 하지만 BPEL4WS의 ALCD에서는 이러한 구별이 없이 모든 activity의 최종적인 state는 "Ended"가 된다. 이러한 모호한 "Ended" state의 정의는 많은 분야에 활용되는 logging의 효율적인 운영에 치명적인 결점을 안겨다 준다.

Active→Canceling→Ended의 과정을 거치거나 Active→Completed→Compensating→Ended의 과정을 거쳐 비성공적으로 종료된 경우에도 모두 "Ended"로 logging이 되어 버림으로써, log 내에서 성공적으로 종료된 activity와 구별할 수 없다. 비성공적으로 종료된 activity를 선택적으로 다시 시도하거나 다른 경로로 재시작하는 경우, 이러한 logging 전략은 결정적인 약점이 되는 것이다.

두 번째는 state들의 의미적인 불일치의 문제이다. Figure 1에 그려진 대로 모든 cancel의 과정은 Canceling→Ended로, close의 과정은 Closing→Ended로 compensate과정은 Compensating→Ended로 표현되어 있다. 모두 진행 중의 state을 나타내는 Canceling, Closing, Compensating과 완료 후의 state를 나타내는 Ended로 구분되어 있다. 하지만,

유독 complete의 과정은 진행 중임을 나타내는 Completing의 state가 생략되어 있어 의미적인 불일치가 생긴다.

3.2 Modified activity life cycle diagram

위에서 언급한 문제점들은 다음과 같이 개선되었다. 첫 번째 문제인 동일한 state로 끝난다는 단점은 "Ended"의 state를 cancel 완료 상태를 나타내는 Canceled state, compensate의 완료 상태를 나타내는 Compensated state, 그리고 close의 완료상태를 나타내는 Closed state로 분화함으로써 해결한다. 두 번째 문제인 의미적인 불일치는 Completing state를 추가함으로써 해결한다. 해결 방안에 따른 최종적으로 수정된 ALCD는 다음의 figure 2에 나타내었다.

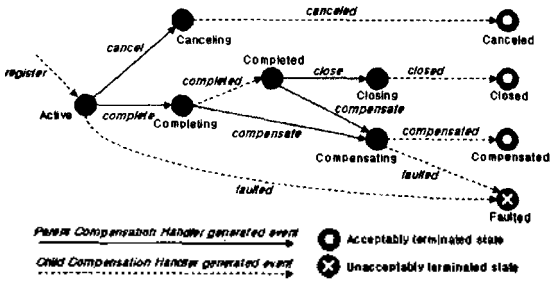

Figure 2. Modified activity life cycle diagram

4. Backward recovery architecture

4.1 Backward recovery architecture

4.1.1 Backward recovery가 필요한 Scope 내의 모든 simple 및 complex web services에 compensation handler들을 부여한다.

Figure 3과 같이 exception handler가 부여된 scope내의 모든 parallel scope, sequence scope, 그리고 simple web services에 compensation handler를 추가한다. Simple web services에는 elementary compensation handler(ECH)를, parallel type의 scope에는 parallel compensation handler(PCH)를 sequence scope에는 sequence compensation handler(SCH)를 부여한다. 예를 들어 figure 3의 (a)의 경우, exception handler가 있는 sequence type의 scope에는 SCH#1을 추가하고, 그 아래의 simple web service A에는 ECH#1를 부여한다.

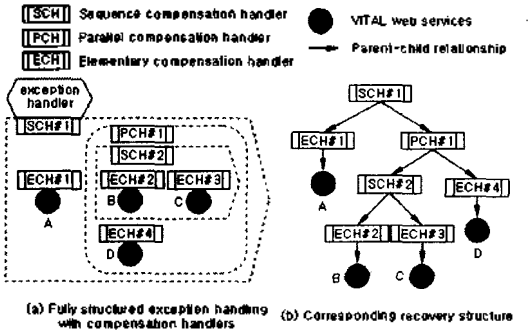

Figure 3. Hierarchical recovery structure의 예

이런 방법으로 모든 scope과 simple web services에 compensation handler들을 부여하면, 총 1개의 PCH와 2개의 SCH, 4개의 ECH가 scope내에 존재하게 된다. Figure 3의 (b)에는 exception handler를 가진 (a)의 scope에 대한 compensation handlers들을 hierarchical recovery structure로 표현했다. 이러한 hierarchical recovery structure에서는 parent child relationship이 동시에 표현된다. SCH#1은 ECH#1과 PCH#1의 parent compensation handler가 되며, PCH#1는 SCH#1과 ECH#4의 parent compensation handler가 된다. 따라서 모든 parent child relationship을 구성해 보면, figure 3의 (b)에서는 총 10개의 parent child relationship이 정의된다.

4.1.2 각 compensation handler에 COMPLETE, COMPENSATE, CLOSE, CANCEL의 4C methods를 부여한다

Compensation handler는 또 다른 웹 서비스가 되며, figure 4과 같이 모든 type의 compensation handler에는 같은 이름의 COMPLETE, COMPENSATE, CLOSE,

CANCEL method(4C methods)를 추가한다.

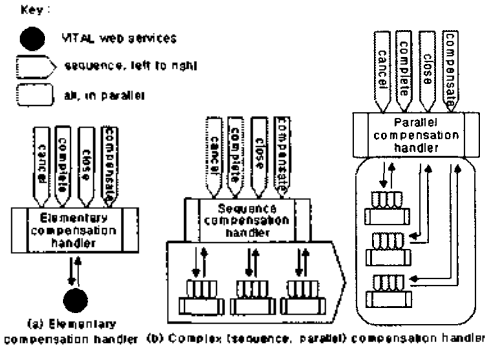

Figure 4. Compensation handlers with 4C methods

Compensation handler의 type에 관계없이 동일한 이름의 4C methods를 가지게 되며, 이러한 작업을 통해 parent compensation handler는 child compensation handler의 type을 고려할 필요 없이 4C methods의 logic을 다음의 section 4.2에서 소개된 바와 같이 작성할 수 있게 된다. 즉, figure 5와 같이 복잡한 1개의 compensation 문제를 여러 개의 단순한 compensation 문제들로 나누어 해결할 수 있는 divide & conquer 방법이 적용 가능한 backward recovery structure가 구성된다.

4.2 4C methods의 구현

4.3.1 ECH의 4C methods

Simple web service X에 elementary compensation handler Y가 부여되어 있으며 X에서는 그들의 local failure를 다룰 수 있는 compensation logic을 제공한다고 가정하자.

(1) COMPLETE: parent인 ECH의 state를 Completing으로 변환한 후 실제 업무 수행 method를 호출한다.

(2) COMPENSATE: ECH의 state를 Compensating으로 바꾸고, 이미 complete된 simple web services X의 compensation method를 호출한다.

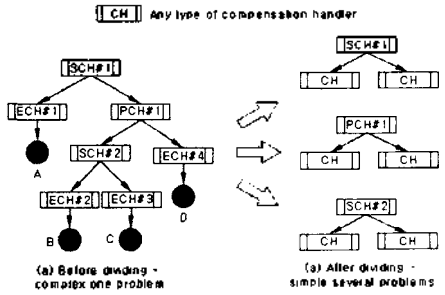

Figure 5. Divide and conquer approach

(3) CLOSE: scope 차원에서 모든 웹 서비스들이 completed되었을 때, 최상위 parent compensation handler가 모든 child에 대해 CLOSE method를 호출한다. 이때 ECH에서는 transaction을 성공적으로 종료한다.

(4) CANCEL: scope 차원에서 중지했을 때 아직 X의 COMPLETE method를 호출하지 않은 경우 state를 Canceling과 Canceled로 순차적으로 변화시킴으로써 X.COMPLETE가 수행되지 않았고 더 이상 수행하려고 시도하지 않겠다는 의사를 표시한다.

4.3.2 Complex compensation handler(SCH or PCH)의 4C methods

PCH나 SCH의 경우에는 ECH와는 달리 1개 또는 그 이상의 child compensation handlers를 가지게 된다.

(1) COMPLETE: compensation handler Y가 PCH인지 SCH인지에 따라 동시에 또는 순서대로 child compensation handlers에 대한 COMPLETE method를 호출한다.

(2) COMPENSATE: Y가 PCH인지 SCH인지에 따라 COMPLETE의 동시에 또는 역순으로 child compensation handlers에 대한 COMPLETE method를 호출한다. 모든 Completed state인 child compensation handlers는 COMPENSATE method를 호출하고, 모든 Active state의 child compensation handler들은

CANCEL method를 호출한다.

(3) CLOSE: 모든 Completed state의 child compensation handlers에 대해 동시에 CLOSE method를 호출한다.

(4) CANCEL: 모든 Active state의 child compensation handlers에 대해 CANCEL method를 호출한다.

5. Discussion & Conclusion

기존 연구들의 가장 큰 문제는 '어떻게 복잡한 비즈니스 프로세스의 올바르고 신뢰할 수 있는 실행을 보장할 것인가?'에 대한 구체적인 방안을 제시하지 못하고 있다는 것이었다. 따라서 본 연구에서는 이러한 문제에 대한 체계적인 접근 방안으로 BPEL4WS의 pi-calculus를 기반으로 한 structured exception handing의 구조하에서 개선된 ALCD와 hierarchical backward recovery structure를 제시하였다. 또한 이들을 활용한 구체적인 state처리와 method 호출에 대해서 설명하였다.

본 연구를 통한 가장 큰 장점은 비즈니스 프로세스의 흐름의 복잡성이 증가하여도 기존과는 달리 backward recovery에 대한 복잡도가 크게 증가 하지 않는다는 것이다. 이러한 장점은 복잡한 하나의 문제를 divide and conquer 전략을 통해 단순화하여 해결 방안을 제시한 것에 기인한다. 요컨데, 본 연구는 해결하기 힘든 문제였던 backward recovery의 구체적인 구현 방안을 간결하고 체계적으로 제시한 것이다.

이 후의 연구는 이미 제시한 complex compensation handler인 PCH와 SCH의 logic을 확장하여 while, switch, OR, XOR 등에 대한 complex compensation handler를 고안하는 방향으로 발전될 것이다.

6. References

[1] Francisco Curbera, Rania Khalaf, Frank Leymann and Sanjiva Weerawarana , "Exception Handling in the BPEL4WS Language", Conference on Business Process Management on the Application of Formal Methods to Process-Aware Information Systems , Eindhoven, The Netherlands, pp 276-290, June 2003.

[2] Howard Smith, Douglas Neal, Lynette Ferrara and Francis Hayden, "The Emergence of Business Process Management", Computer Science Corporation (CSC) Research Services, January 2002.

[3] Johann Eder and Walter Liebhart, "Workflow Recovery", Conference on Cooperative Information Systems, pp.124-134, 1996.

[4] Robin Milner, "Communicating and Mobile Systems: The π-Calculus", Cambridge University Press, 1999.

[5] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Ivana Trickovic, and Sanjiva Weerawarana, "Business Process Execution Language for Web Services (BPEL4WS) 1.1", published on the world wide web by BEA Corp., IBM Corp., and Microsoft Corp. at http://www-106.ibm.com/developerworks/library/ws-bpel/, May 2003.

[6] Won Kim, "Modern Database Systems: The Object Model, Interoperability, and Beyond", Addison Wesley Professional, pp 592-620, 1995.