시스템의 가용성을 지원하는 소프트웨어 개발 프로세스

정효택 · 주상현 (한국전자통신연구원)

Ⅰ. 서 론

요구공학(requirements engineering)이 발전함에 따라 소프트웨어의 기능적 요구사항(functional requirements)과 함께 비기능적 요구사항(non-functional requirements)을 지원하는 소프트웨어 개발 프로세스들에 대한 연구가 진행되어 왔다.

일반적인 소프트웨어 개발 프로세스들은 선형(linear), 비선형(non-linear), 나선형(spiral), 진화형(incremental) 등의 형태로 제안되었다. 선형모델로는 I. Sommerville의 반복적인 액티비티(iterative activities)를 허용하는 개념적 선형 모델[1]과 L. A. Macaulay의 순수 선형 모델[2]이 제안되었으며, 비선형 또는 나선형 모델로는 P. Loucopoulos의 반복적이고 주기적인 모델[3]과 B. W. Boehm 의 나선형 모델[4]이 제안되었다.

K. Sousa는 Upi 모델[5]을 제안하였는데, 이 모델은 소프트웨어 개발시에 사용성(usability)을 제공하기 위하여 HCI (Human Computer Interaction)개념을 도입한 사용자 인터페이스를 제공한다. Upi 프로세스는 IBM의 RUP (Rational Unified Process)와 통합하여 개발 환경에 따라 커스터마이징(customizing) 될 수 있다. S. Rottger는 소프트웨어 개발 초기 단계에서 추상적인 비기능적 요구사항을 명세하고 정제함으로써, 비기능적 요구사항의 제한점들을 표현할 수 있는 모델[6]을 제안하였다.

현재까지 공학자들이 해결하려고 노력하는 이슈들 중의 하나는 비기능적 요구사항을 개발 프로세스의 어느 부분에서 처리해야 하는가 하는 것이다. 예를 들면 운용 시스템(operational system)의 비기능적 요구사항은 프로세스 자체 혹은 프로덕트 등과 관련된 일종의 제한사항으로 간주되고 있다. V. Sivess는 의사 결정시에 CCS (Calculus of Communication Systems)를 사용하는 모델링 접근법을 제안하였다[7]. CCS 표기법은 서로 다른 다큐먼트 상태(document states)에서 각각의 역할 에이전트(role agents)들에 의해 수행되는 다양한 활동들을 표현하는데 적합하다. Pi-calculus는 활동들 사이의 새로운 추적관계나 제한사항(constraints)이 발생할 경우 이를 모델링하는데 사용된다. V. Sivess가 제안한 모델은 정형명세를 사용하기 때문에 복잡한 다큐먼트 상태에 종속적인 일련의 프로세스들을 매우 자연스럽게 표현할 수 있다. 또한 이 모델은 소프트웨어 프로세스내에서 비기능적 요구사항을 처리함으로써 저비용으로 소프트웨어의 마이그레이션(migration)을 지원하는 일반적인 가이드라인을 제공한다. 하지만 정형명세를 사용하기 때문에 지원도구 없이 프로세스를 사용하기가 수월하지 않은 단점이 있다.

본 논문의 구성은 다음과 같다. 제2장에서는 일반적인 가용성에 대해서 알아보고, 제Ⅲ장에서는 가용성(availability)을 지원하는 소프트웨어 개발 프로세스를 제안한다. 이 프로세스는 성능(performance) 및 보안(security) 같은 비기능적 요구사항도 지원 가능하다[8,9,10]. 제Ⅳ장에서는 시뮬레이션을 통하여 제안한 프로세스의 단계적인 검증을 시도하였다. 마지막으로 제Ⅴ장에서는 향후 연구 방향과 결론을 내린다.

Ⅱ. 시스템의 가용성(availability)

시스템의 가용성은 비즈니스에 직접 영향을 미치기 때문에 사용자는 물론 경영층에서 가장 관심이 집중되는 비기능적 요소 중의 하나이다. 시스템은 항상 가용하여야 하며 그렇지 않을 경우 대체 기능을 제공하여야 한다. 예를 들면 온라인 시스템에서 네트워크상에 문제가 발생하여 시스템이 가용

하지 않은 경우, 오프라인 모드로 변경할 수 있는 기능이 제공되어야 한다.

시스템의 가용성은 시스템 백업과 복구가 크게 좌우한다. 즉 결함(fault) 발생시 적정 시간내에 시스템이 백업되고 복구가 되어야 한다. 이러한 백업과 복구를 위해서 발생 가능한 결함을 정의하고 이를 예방하거나 처리하는 프로세스와 매커니즘이 필요하다.

가용성은 다음과 같이 측정된다.

$$\text{가용성} = \frac{\text{시스템이 가용한 시간}}{\text{경과 시간}}$$

만약 하루에 23시간이 시스템이 가용하다면, 그 시스템의 가용성은 95.83%가 된다 (23/24=0.9583).

반면에 비가용성(unavailability)는 다음과 같이 측정된다.

$$\text{비가용성} = \frac{\text{시스템이 가용하지 않은 시간}}{\text{경과 시간}}$$

또한 $\text{가용성} + \text{비가용성} = 100\%$이다.

시스템의 가용성은 시스템이 가용한 확률로 생각할 수 있다. 즉, 고장간의 평균시간 (MTBF :Mean-Time-Between-Failure), 고장이 발생하는 평균시간, 즉 시스템이 정상 작동하는 평균시간 (MTTF : Mean-Time-To- Failure), 보수하는데 걸리는 평균시간 (MTTR : Mean-Time-To-Repair) 의 조합으로 표현할 수 있다.

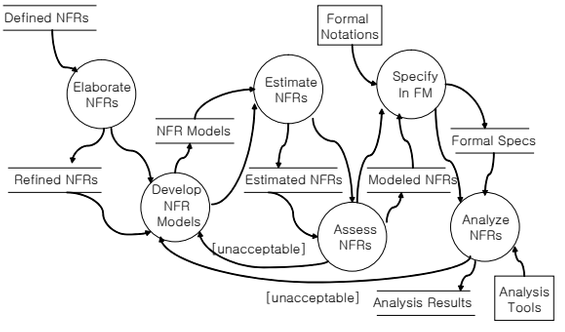
$$\text{가용성} = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

MTTR는 가용성 혹은 유지보수성을 측정하는데 가용한 비기능적 요소이며, 정상적인 작동에 소요되는 평균 시간을 말한다. 그러므로 시스템의 가용성을 위해서는 여러 가지 결함에 따르는 MTTR의 예측이 필요하다. 이러한 예측은 시스템 설계자는 보수를 담당하는 전문기술자들의 경험에 근거하여 이루어진다.

일반적으로 하드웨어의 MTTR은 결함이 있는 컴퓨터의 교체나 보수에 걸리는 시간을 의미하며, 소프트웨어의 MTTR은 모듈에서의 결함이 발견되고 부터 재부팅되는데 걸리는 시간을 의미한다. 소프트웨어를 보수하는데 걸리는 시간을 측정, 그룹핑하여 일반화하기는 용이하지 않다. 소프트웨어의 보수는 종종 소프트웨어의 재배치(redistribution, redeployment)를 포함하는 등 단순하지 않으며, 일단 모듈이 보수가 되었더라도 결함 분석, 테스팅, 재배치, 손상된 데이터 복구 등의 일련의 프로세스가 필요하다.

Ⅲ. 가용성을 지원하는 소프트웨어 개발 프로세스

본 장에서는 가용성을 지원하는 소프트웨어 개발 프로세스를 제안한다. 제안하는 프로세스는 가용성을 가공(Elaborating), 모델 개발(Developing Model), 추정(Estimating), 평가(Assessing), 정형명세(Specifying in Formal Method), 분석(Analyzing)하는 여섯 단계로 구성되어 있다. 제안하는 프로세스는 가용성 뿐만 아니라 성능, 보안 등과 같은 비기능적 요구사항에도 적용 가능하다. 〈그림 1〉은 제안한 프로세스의 전체 구조를 데이터 흐름 다이어그램(Data Flow Diagram, DFD)을 이용하여 보여주고 있다.



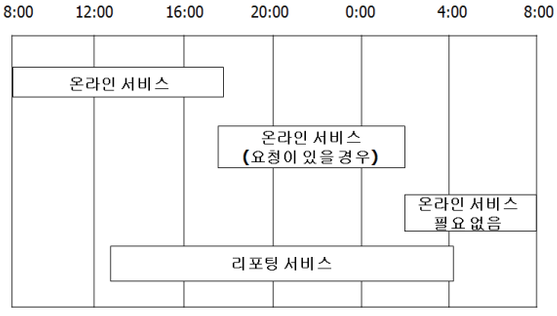
〈그림 1〉 프로세스 전체 구조도

1. 1단계: 가공

가공 단계에서는 비기능적 요구사항을 도출, 확정, 정제하는 활동들로 구성된다. 사용자의 요구(needs)나 개략적인 요구사항 명세서(Software Requirements Specification, SRS)가 입력되고, 결과로는 정제된 요구사항 명세서(Refined SRS)가 작성된다. 가용성의 경우 제공되는 서비스를 정제하거나 타입에 따라 분류하거나 서비스의 레벨을 정의한다.

2. 2단계: 모델 개발

비기능적 요구사항 모델 개발 단계에서는 정제된 요구사항으로부터 적절한 매트릭스나 모델을 개발하는 활동들로 구성된다. 전단계에서 작성된 정제된 요구사항 명세서가 입력되거나 결과로는 모델이나 매트릭스가 개발된다. 요구사항 평가 혹은 분석 단계에서 그 결과가 수락되지 않을 경우 재실행된다. 가용성의 경우 가용한 자원을 식별하고 정상적인 가용성 모델(availability model)을 만든다. 가용하지 않은 자원을 식별하고 비가용성 모델(unavailability model)을 만든다. 〈그림 2〉는 시스템이 가용한 시간과 그렇지 않은 시간을 간트차트를 이용하여 나타낸 예이다.



〈그림 2〉 간트차트를 이용한 가용성 모델

3. 3단계: 추정

비기능적 요구사항 추정 단계에서는 전단계에서 생성된 모델이나 메트릭스를 측정, 추정, 테스팅하는 활동들로 구성된다. 전단계에서 생성된 모델이나 메트릭스가 입력되고, 비기능적 요구사항의 추정값들이 생성된다. 가용성의 경우 플랫폼의 가용성을 추정하고 기능적 가용성을 설계한다. 〈그림 3〉에서는 플랫폼의 가용성확보를 위한 고장 처리 시나리오의 예를 보여주고 있다.

현상	영향	처리 요령	처리시간
하드웨어 고장 (디스크고장은 제외)	유용성 감소	고장이 발생한 부품을 교체 H/W나 OS의 Reconfiguration	1 시간
디스크 고장	유용하지 않음 (unavailability)	고장난 디스크 교체 데이터 백업	6 시간
OS 충돌	일시적 서비스 장애	리부팅	10분
...			
...			

〈그림 3〉 고장 처리 시나리오

4. 4단계: 평가

비기능적 요구사항 평가 단계에서는 SRS에 명기되어 있는 비기능적 요구사항이나 혹은 설계 모델에 명기되어 있는 비기능적 요구사항을 평가하는 활동들로 구성된다. 전단계에서 생성된

추정값들이 입력되고, 평가결과들이 산출된다. 평가결과가 수락되지 않으면, 수락될 때 까지 2, 3 단계의 작업들이 반복된다. 가용성의 경우 플랫폼과 기능적 가용성을 통합하고, 위험 요소를 식별하고 그 해법을 찾으며, 관련된 요구사항과 소프트웨어 아키텍처의 일관성을 점검한다.

5. 5단계: 정형명세

비기능적 요구사항 정형명세 단계에서는 전단계에서 생성된 비기능적 요구사항을 정형기법을 사용하여 명세하는 활동으로 구성된다. 비기능적 요구사항은 객체의 행위(behavior), 구조 및 상호 인터랙션을 표현할 수 있는 UML (Unified Modeling Language) 모델로 표현이 가능하다. Petri Nets, LOTOS, Z-notation, CPS(Communicating Sequential Process), CCS 등의 기법들이 UML 모델을 정형명세 하는데 사용될 수 있다. 입력은 비기능적 요구사항이 표현된 소프트웨어 아키텍처이며, 결과물은 정형명세이다.

5. 6단계: 분석

마지막 단계인 비기능적 요구사항 분석 단계에서는 정형명세된 비기능적 요구사항을 분석한다. UML 모델에 명기된 비기능적 요구사항들 을 시뮬레이션하고 그 결과를 분석한다. 만약 분석된 자료가 수락이 되지 않으면, 수락될 때 까지 2단계 이후의 활동들을 반복 수행한다. 입력은 정형명세된 비기능적 요구사항이고, 결과물은 시뮬레이션 결과를 분석한 정보이다.

Ⅳ. 가용성 측정을 위한 시뮬레이션

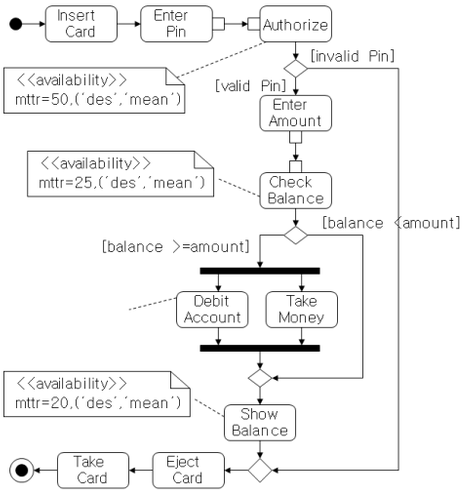
본 장에서는 제3장에서 제안한 프로세스 중 비기능적 요구사항을 정형명세하고 분석하는 5, 6단계를 간단한 현금자동지급기 (ATM) 응용 프로그램에 적용함으로써, 그 타당성을 검증한다.

〈그림 4〉와 같이 ATM 기능을 UML 액티비티 다이어그램으로 모델링하였으며, 가용성 측정을 위하여 Authorize와 CheckBalance 모듈의 MTTR을 각각 50, 25 UT(Unit Time)로 가정하였다.

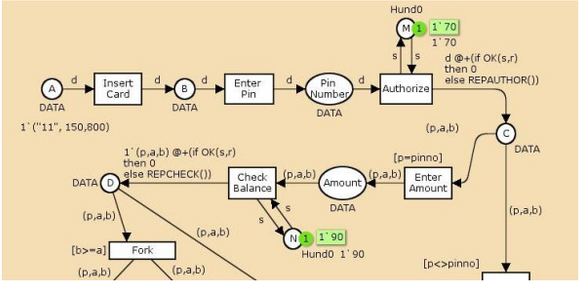
모델링 선행 작업으로 UML 액티비티 모델을 Colored Petri Net (CPN) 으로 변환하는 매핑룰과 알고리즘을 개발하였으며, 이를 활용하여 〈그림 4〉의 액티비티 모델을 〈그림 5〉와 같은 CPN(Colored Petri Net) 모델로 변환하였다. 〈그림 5〉에서 Authorize와 CheckBalance 모델에서 정상 실행할 확률을 각각 0.7과 0.9로 가정하였음을 알 수 있다.

CheckBalance 모듈의 정상적인 실행확률을 0.7, 0.8, 0.85, 0.9로 변경하면서 200번의 예금 인출 시뮬레이션 수행한 결과를 〈표 1〉에서 보여주고 있다.

CheckBalance와 Authorize가 정상실행을 할 확률이 각각 0.7, 0.9일 경우, MTTR이 75UT 일 경우 5번이 발생하였고, MTTR이 0UT일 경우 14번이 발생하였음을 알 수 있다. 여기서 MTTR이 0UT이라 함은 정상적으로 예금이 인출되었음을 말하며, 25UT는 CheckBalance에서 결함이 발생하였음을



〈그림 4〉 MTTR을 포함한 액티비티 모델



〈그림 5〉 MTTR 시뮬레이션을 위한 CPN 모델

〈표 1〉 시뮬레이션 결과

MTTR (UT)	CheckBlance 정상실행 확률			
	0.70	0.80	0.85	0.9
0	120	134	153	163
25	14	17	18	14
50	61	45	27	20
75	5	4	2	3

의미한다. 50UT는 Authorize에서 결함이 발생하였고, 75UT은 CheckBalance와 Authorize 모듈 모두에서 결함이 발생하였음을 의미한다.

시뮬레이션 결과를 분석하여 제안한 프로세스의 타당성을 검증하기 위해 본 실험에서는 퍼센티지 에러(Percentage Error, PE)를 이용한다. PE는 실제적으로 측정한 값이 이론적으로 추정한 값과 어느 정도 차이가 나는지를 나타내는 측정 지표가 되는데, 다음과 같이 계산된다.

$$Percentage\ Error = \frac{|\text{추정값} - \text{측정값}|}{\text{측정값}} * 100$$

각 모듈의 MTTR에 대한 추정값과 실제 측정값은 다음과 같이 계산된다.

$$\text{추정값} = \sum_{n=1}^{k} \frac{\sum \text{실패확률} * MTTR}{\sum \text{실패확률}} \ (k: \text{모듈수})$$

$$\text{측정값} = \frac{\sum_{n=1}^{k} \text{실패한 횟수} * MTTR}{\text{실패한 횟수}} \ (k: \text{모듈수})$$

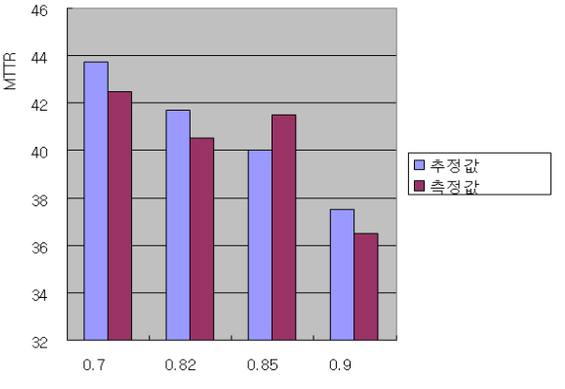
이 식에 의하면 CheckBalance의 성공적인 실행확률이 0.7인 (즉, 실패확률이 0.3) 경우

$$\text{추정값} = (\frac{0.1}{0.1+0.3})(25) + (\frac{0.1}{0.1+0.3})(50) = 43.75$$

$$\text{측정값} = \frac{(19)(25)+(66)(50)-(5)(75)}{200-120} = 42.50$$

이 된다.

〈그림 6〉에서는 CheckBalance의 성공적인 실행확률이 0.7, 0.8, 0.85, 0.9 일 경우의 추정값과 측정값을 비교하였으며, 〈표 2〉에서 PE 〈 3.6%임을 알 수 있다.



〈그림 6〉 Case별 추정값과 측정값의 비교

〈표 2〉 MTTR을 포함한 액티비티 모델

	CheckBlance 정상실행 확률			
	0.70	0.80	0.85	0.9
추정값	43.75	41.67	40.00	37.50
측정값	42.50	40.53	41.49	36.49
PE(%)	2.94	2.81	3.59	2.77

V. 결 론

본 논문에서는 가용성 등과 같은 비기능적 요구사항이 포함된 소프트웨어 개발을 지원하는 소프트웨어 개발 프로세스를 제안하였다. 이 프로세스는 6단계로 구성되어 있으며, 각

단계에서 수행되어야 하는 세부적인 활동들이 정의되어 있다. 특정 비기능적 요구사항을 지원하기 위해서는 요구사항의 특성에 따라 활동들이 카스터마이징 될 수 있다.

제안한 소프트웨어 프로세스의 타당성을 검증하기 위하여 가용성의 척도가 될 수 있는 MTTR과 같은 비기능적 요구사항을 포함한 모델을 분석하였다. 즉, UML모델을 CPN 모델로 변환하고, 그 모델을 대상으로 시뮬레이션과 결과 분석의 방법을 설명함으로써, 프로세스의 5, 6 단계의 타당성을 보여주었다.

향후 계획으로는 제안된 프로세스의 1단계부터 4단계까지의 활동들에 대한 타당성 검증을 준비하고 있으며, CPN 모델뿐만 아니라 다른 다양한 정형명세 기법으로 표현된 모델에 대한 적용도 고려하고 있다. 또한 성능이나 보안과 관련된 비기능적 요구사항을 포함하는 모델에 대해서도 제안된 프로세스를 적용해 볼 계획을 갖고 있다.

참 고 문 헌

[1] I. Sommerville and G. Kotonya, Requirements Engineering: Processes and Techniques, John Wiley and Sons Ltd, 1998.

[2] L. A. Macaulay, Requirements Engineering, Springer-Verlag, 1996.

[3] P. Loucopoulos and V. Karakostas, System Requirements Engineering, McGraw-Hill, 1995.

[4] B. W. Boehm, A Spiral Model of Software Development and Enhancement, Computer, vol. 21, no. 5, pp. 61-72, 1988.

[5] K. Sousa, E. Furtado, and H. Mendonca, UPi- A Software Development Process Aiming at Usability, Productivity and Integration, Proceedings of the 2nd Latin American Conference on Human Computer Interaction (CLIHC '05), pp. 76-87, Cuernavaca, Mexico, October 2005.

[6] S. Rottger and S. Zschaler, A Software Development Process Supporting Non-Functional Properties, Proceedings of the IASTED International Conference on Software Engineering, Innsbruck, Austria, February 2004.

[7] V. Sivess, Non-Functional Requirements in the Software Development Process, Software Quality Journal, vol. 5, no. 4, pp. 285-294, December 1996.

[8] 정효택, 주상현, 비기능적 요구사항을 지원하는 소프트웨어 개발 프로세스, 정보통신설비학회논문지 제9권 제1호, pp 13-18, 2010.3.

[9] H. Jung and S. Joo, Transformation of an Activity Model into a Colored Petri Net Model, Proceedings of the 2nd International Conference on Trends in Information Sciences & Computing 2010, Chennai, India, pp. 32-37, December 17-19, 2010.

[10] H. Jung and G. Lee, A Systematic Software Development Process for Non-Functional Requirements, Proceedings of the International Conference on ICT Convergence 2010, pp. 431-436, Juju, Korea, November 17-19, 2010.

정 효 택

1986년 2월 경북대학교 전자공학과 학사
1997년 8월 연세대학교 컴퓨터과학과 석사
2008년 8월 텍사스 주립대(달라스) 컴퓨터과학과 박사
1985년 12월~1987년 12월 삼성전관(현 삼성SDI) CE부
1987년 12월~1996년 4월 시스템공학연구소
1996년 4월~현재 한국전자통신연구원
〈관심분야〉 S/W 품질보증, S/W 아키텍처, S/W 프로세스, 요구공학

주 상 현

1989년 2월 동국대학교 전자공학과 학사
1994년 2월 동국대학교 전자공학과 석사
1999년 3월 국립니기타대학 자연과학연구과 박사
1999년 4월~2001년 3월 국립니기타대학 전기전자공학과 조수
2001년 3월~현재 한국전자통신연구원 콘텐츠연구본부 디지털액터 연구팀 책임연구원
〈관심분야〉 실감미디어, MPEG 표준화