Sequence Diagram을 이용한 테스트 오라클 생성

정정수[0] 김형호 조승모 권용래
한국과학기술원 전산학과
{jschung, hhkim, smcho, kwon}@salmosa.kaist.ac.kr

Generating Test Oracles from Sequence Diagram for Java Application

Chung-Su Chung Hyung-Ho Kim Seung-Mo Cho Yong-Rae Kwon
Dept. of Computer Science, KAIST

요 약

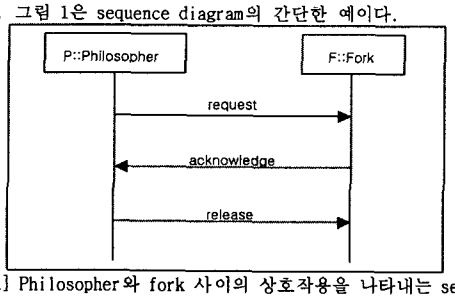
이 논문에서는 산업계의 표준으로 널리 사용되고 있는 객체 지향 시스템의 명세 언어인 UML의 sequence diagram을 이용하여 객체 지향 시스템을 검증하는 방법과 이 방법을 테스팅 오라클을 생성하는 데 사용할 수 있는 프레임웍을 제시하였다.
우리는 sequence diagram을 테스팅 결과의 검증에 사용하기 위하여 정형적으로 재정의 하였다. 그리고 시제 논리의 강력한 검증 능력을 사용하기 위해서 Half-Order Dynamic Temporal Logic(HDTL)이라 불리는 새로운 시제 논리를 정의하였고 sequence diagram을 HDTL 논리식으로 변환시키는 의미 함수(semantic function)를 정의하였다. HDTL에서 오토마톤을 생성하기 위해서 Tableau 방법을 변형하여 적용시켰다. 이 결과 생성된 오토마톤은 이상 상태(anomaly), 즉 sequence diagram에 표현되지 않은 사건(event)의 발생을 검색하는 오라클로 사용할 수 있다. 테스팅의 결과를 수작업으로 검증하는 것은 매우 어렵고 오류가 발생하기 쉬운 작업이므로 제안한 방법은 유용하게 사용될 수 있다.

1. 서론

객체 지향 기술은 응용 프로그램 개발에 널리 받아들여지고 있는 기술로 정확하고 명확하게 프로그램을 모델링할 수 있는 여러 개념들을 제공한다. 이러한 강력한 개념을 지원하는 객체 지향 방법론에서도 테스팅과 같은 검증 작업을 수행하는 데는 많은 어려움이 있다. 이는 객체 지향 시스템들의 유연한(flexible) 행위가 검증 작업을 어렵게 만들기 때문이다.
객체들은 과거 의존적인(history dependent) 행위를 하기 때문에 기존의 절차 지향적 시스템(procedural system)에서 사용하던 검증 기법을 객체 지향 시스템에 적용하기 어렵다. 이런 객체들의 행위는 반응형 시스템(reactive system)과 같이 trace라 불리는 사건의 연속으로 생각할 수 있다. 기존의 여러 반응형 시스템의 검증 방법들이 객체 지향 시스템에 적용되었는데 예를 들어 FSM(finite state machine)에 기반을 둔 방법[1]과 페트리 넷(Petri net)에 기반을 둔 접근법[2] 등이 있었다. 하지만 이런 여러 연구들은 그 바탕을 이루는 수학적인 정형 기법의 어려움이나 그 명세 기법이 객체 지향 기법에서 잘 사용되지 않는 이유로 인해 실제 산업계에서 거의 사용되지 않고 있다.
이 논문에서는 강력한 정형 기법인 시제 논리(temporal logic)를 산업계의 표준으로 널리 사용되고 있는 객체 지향 시스템의 명세 언어인 UML의 sequence diagram에 적용하여 기존 연구들이 가지고 있는 문제들을 극복하고자 한다. Sequence diagram은 객체들 간의 특정한 시나리오를 표현하는 명세 방법으로 간단하고 직관적인 의미 구조로 객체 지향 시스템이나 통신 시스템과 같이 객체들이 주고 받는 영향이 중요한 의미를 가지는 곳에 널리 사용되고 있다. 특히 OBJECTORY[7]나 RUP(Rational Unified Process)[3]과 같은 여러 객체 지향 기법들은 sequence diagram이나 그 변형을 표준 명세 기법으로 사용하고 있다. 따라서 sequence diagram은 정형 기법을 적용하는 대상으로 적절하다 할 수 있다.
시제 논리는 반응형 시스템을 기술하고 검증하는 데 가장 널리 사용되는 방법 중의 하나이다. 시스템을 기술하고 있는 시제 논리식은 Tableau 방법을 사용하여 오토마톤으로 변형될 수 있다. 이 방법은 시제 논리식이 사건의 연속에 대해서 성립하는지를 판단하는 대신 오토마톤이 사건의 연속, 즉 trace를 받아들이는지 판단하는 문제로 바꾸어 준다. 오토마톤의 언어 수용 문제는 선형적인 시간에 해결할 수 있는 문제로 효과적인 해결이 가능하다.
우리는 고전적인 일차 한정사인 All(∀)과 Exist(∃) 대신에 Half-Order Temporal Logic[6]에서 제안된 동결 한정사(freeze quantifier)를 사용하였다. 고전적인 한정사 All(∀)과 Exist(∃)는 한정사에 의해 도입된 변수들의 의미를 집합에 대해서 정의하고 사용하기 때문에 정형 기법을 효과적으로 다루는데 어려움이 있다. 이에 반해 동결 한정사는 사건의 연속인 trace에 대해서만 정의되기 때문에 시제 논리식을 집합 표현을 사용하지 않고 다룰 수 있게 해준다. 이 논문에서는 sequence diagram을 표현하는 시제 논리로서 Half-Order Temporal Logic을 변형시킨 Half-Order Dynamic Temporal Logic(HDTL)을 제안하고 정의하였다. 그리고 sequence diagram을 HDTL로 변환시켜 주는 의미 함수를 정의하고 이 시제 논리식을 오토마톤으로 변형시키는 Tableau[4] 방법에 대해서 설명과 예를 들었다.
테스팅은 여러 데이터를 반복해서 수행하고 그 결과를 명세와 비교해 보는 검증 과정이 필요하다. 이런 여러 번의 검증 작업을 수작업으로 수행하는 것은 어려운 일이고 오류가 발생할 확률이 높다. 이 논문에서는 앞서 설명한 방법을 자동화하여 객체 지향 시스템의 명세로부터 테스트 오라클을 생성할 수 있는 프레임웍을 제안하고 있다. 이렇게 생성된 오라클을 테스트 자동화에 이용하면 수작업을 이용한 검증을 최소화할 수 있다.
이 논문의 구성은 다음과 같다. 2장에서 sequence diagram을 예를 들어 설명하고 그에 대한 정형적인 정의를 하였다. 3장에서는 HDTL의 문법과 의미를 정의하였다. 4장에서 의미 함수와 Tableau 방법에 대해서 설명하였다. 5장에서는 테스팅에 적용할 수 있는 프레임웍을 설명하였다. 6장에서는 결론과 함께 향후 연구 과제에 대해서 설명하였다.

2. Sequence Diagram

Sequence diagram은 객체지향 시스템을 분석 설계하는데 산업계의 표준으로 널리 사용되고 있는 UML의 일부이다. Sequence diagram은 어떻게 객체들이 서로 영향을 주고받는지를 표현하고 있으며 특히 메시지의 순서 관계를 표현하는데 많이 사용된다. Sequence diagram은 두 개의 축으로 이루어진다. 수직의 축은 시간의 흐름을 표현하고 수평의 축은 그 사건 순서(event sequence)에 관여하는 객체들의 집합을 나타낸다. 수직 축 사이의 점패선은 객체들이 주고 받는 메시지를 표현한다. 그림 1은 sequence diagram의 간단한 예이다.



[그림 1] Philosopher와 fork 사이의 상호작용을 나타내는 sequence diagram

sequence diagram을 테스트 오라클을 생성하는데 사용하기 위하여 정형적으로 정의하였다. 편의를 위해서 sequence diagram의 특성 중 일부에 대해서만 정형적인 정의를 하였다. 그 하나의 예로 sequence diagram 상의 모든 메시지를 동기적으로 해석하였다.

Definition 1 (Sequence diagram) Sequence diagram S는 다음과 같은 요소들을 지닌 레이블이 붙은 방향성 비순환 그래프(labeled directed acyclic graph)이다.

Roles : 역할(role)들의 유한 집합 R

Message : 메시지들의 유한 집합 M

Message Labels : M 에 속하는 각 메시지를 삼중식(triple) (s,r,l)로 대응시키는 함수 g이다. 여기서 l은 메시지의 레이블을 나타내고 s와 r은 R에 속하는 role을 표현하며 발신자, 수신자로 읽는다. l(m) 은 g(m) 의 레이블을 나타내며 s(m)과 r(m)은 그 발신자와 수신자를 나타낸다. Role r 에 속하는 메시지의 집합을 M_r로 표현한다. 즉, $M_r = \{ m \mid m \in M, s(m) = r \vee r(m) = r\}$이다.

Visual Order : 집합 M에 속하는 메시지 사이에는 부분 순서 관계(partial order) < 가 존재한다. 이 < 관계는 M_r에 속하는 메시지 사이의 지역적 전체 순서 관계(local total order) $<_r$로부터 다음과 같이 유도된다.

$m < m'$ 은 $m <_r m'$ 을 만족하는 r이 존재할 때 성립한다. 지역적 전체 순서 관계(local total order) $<_r$ 은 sequence diagram에서 메시지가 표현되는 순서에 따른다.

우리는 시스템의 행위를 메시지의 발생을 나타내는 사건(event)의 유한한 연속, 즉 trace 로 간주하였다. 정형적으로 한 사건 $e \in \mathbb{E}$는 <sender, receiver, label>의 삼중식(triple)으로 표현된다. 객체들의 유한 집합을 O, 메시지 레이블(label)들의 유한 집합을 L이라 했을 때 레이블 $l \in L$은 발생한 메시지의 레이블을 나타내며 $l_e(e)$로 표현한다. sender$\in$O, receiver$\in$O는 그 메시지의 발신자와 수신자를 나타내며 $s_e(e)$, $r_e(e)$로 표현한다.

그 외에 특수 메시지인 ∞와 그 발생 사건 ∞_e를 정의하였다. 메시지 ∞는 sequence diagram 의 마지막 메시지를 의미하는 특수 메시지로 어떤 유효한 sequence diagram 이나 trace도 ∞나 ∞_e를 포함하여서는 안된다. 즉, 모든 메시지 m에 대해서 $l(\infty) = l_e(\infty_e)$, $s(m) \neq s(\infty)$, $r(m) \neq r(\infty)$이 성립하고 모든 사건 e 에 대해서 $s_e(e) \neq s_e(\infty_e)$, $r_e(\infty) \neq r_e(\infty_e)$가 성립한다.

이후부터 문맥상 의미가 명확한 경우 아래 첨자 e를 생략할 수 있다.

3. Half-order Dynamic Temporal Logic

여기서는 sequence diagram에 나타나는 시스템의 행위를 논리식으로 표현하는 Half-order Dynamic Temporal Logic(HDTL)을 정의하고 설명하고자 한다.

변수들의 집합을 V라 했을 때 HDTL로 표현되는 논리식은 명제 기호(proposition symbol)와 부울 연결사(Boolean connective), 시제 연결사(*temporal operator*), 동결 한정사(*freeze quantifier*)로 표현되며 그 문법과 의미는 다음과 같다.

Definition 2 (Syntax of HDTL) HDTL의 항(term)과 식(formula)은 다음과 같이 재귀적으로 정의된다.

Term π := $snd(x) \mid rcv(x) \mid label(x) \mid l$

Formula $\phi := \pi_1 = \pi_2 \mid$ false $\mid$ true $\mid \phi_1 \rightarrow \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \odot\phi \mid \Box \phi \mid \Diamond\phi \mid x.\phi$

여기서 x는 V에 l은 L에 속하는 원소이다.

Definition 3 (Semantics of HDTL) σ를 어떤 한 trace라 하고 $\varepsilon : V \rightarrow E$ 을 변수들에 대한 환경(environment) 함수라 하자. 이때 (σ, ε)의 쌍은 $\sigma \vDash_\varepsilon \phi$이 성립할 경우에만(if and only of) HDTL 논리식 ϕ을 만족한다. 여기서 $\vDash$ 관계는 다음과 같이 재귀적으로 정의된다.

$\sigma \vDash_\varepsilon \pi_1 = \pi_2$	iff $\varepsilon(\pi_1) = \varepsilon(\pi_2)$
$\sigma \nvDash_\varepsilon$ false	
$\sigma \vDash_\varepsilon$ true	
$\sigma \vDash_\varepsilon \phi_1 \rightarrow \phi_2$	iff $\sigma \vDash_\varepsilon \phi_1$ implies $\sigma \vDash_\varepsilon \phi_2$
$\sigma \vDash_\varepsilon \phi_1 \wedge \phi_2$	iff $\sigma \vDash_\varepsilon \phi_1 \wedge \sigma \vDash_\varepsilon \phi_2$
$\sigma \vDash_\varepsilon \phi_1 \vee \phi_2$	iff $\sigma \vDash_\varepsilon \phi_1 \vee \sigma \vDash_\varepsilon \phi_2$
$\sigma \vDash_\varepsilon \neg\phi$	iff $\sigma \nvDash_\varepsilon \phi$
$\sigma \vDash_\varepsilon \bigcirc \phi$	iff $\lvert\sigma\rvert > 1 \rightarrow \sigma^1 \vDash_\varepsilon \phi$
$\sigma \vDash_\varepsilon \odot\phi$	iff $\lvert\sigma\rvert > 1 \rightarrow \sigma^1 \vDash_\varepsilon \phi$
$\sigma \vDash_\varepsilon \Box \phi$	iff $\sigma^i \vDash_\varepsilon \phi$ for $\forall i \geq 0$
$\sigma \vDash_\varepsilon \Diamond\phi$	iff $\sigma^i \vDash_\varepsilon \phi$ for $\exists i \geq 0$
$\sigma \vDash_\varepsilon x.\phi$	iff $\sigma \vDash_{\varepsilon[x:=\infty]} \phi$

4. 오라클 생성

4.1 의미 함수(Semantic Function)

sequence diagram에 표현된 정보를 HDTL 논리식으로 변환하는 의미 함수를 그림2에서 정의하였다. 함수 sem의 정의에 사용된 연산자 $\oplus$은 겹침(overloading)을 뜻한다. 의미 함수의 엄격한 정의는 formula 함수에서 시제 연산자인 $\Diamond$을 제거하면 얻어진다.

```
fun sem(R, M, g, <) =
   prev := [∀r ∈ R : r ↦ ⊥];
   traces := {<(v1, m1), ... (vm, mm), (vm+1, ∞)> :
      for ∀<m1, m2, ..., mm> ∈ seq(M)
        such that vi are distinct variables  for 1 ≤ i ≤ m+1};
   return ∀t ∈ traces : ∨ ◇ formula(t);

   where fun formula(M) =
      if M = <(v, m)> then return v.("node(v, m)")
      else return v.("node(M0)" ∧ ⊙◇"formula(M1)")

   and fun node(v, m) =
      if prev(s(m)) = ⊥ ∧ prev(r(m)) = ⊥ then
         prev := prev ⊕ [s(m), r(m) ↦ snd(v), rcv(v)];
         return label(v) = "l(m)"
      else if prev(s(m)) ≠ ⊥ ∧ prev(r(m)) = ⊥ then
         prev := prev ⊕ [r(m) ↦ rcv(v)];
         return label(v) = "l(m)" ∧ snd(v) = "prev(s(m))"
      else if prev(s(m)) = ⊥ ∧ prev(r(m)) ≠ ⊥ then
         prev := prev ⊕ [s(m) ↦ snd(v)];
         return label(v) = "l(m)" ∧ rcv(v) = "prev(r(m))"
      else (* prev(s(m)) ≠ ⊥ ∧ prev(r(m)) ≠ ⊥ *)
         return label(v) = "l(m)" ∧ snd(v) = "prev(s(m))"
           ∧ rcv(v) = " prev(r(m))"
```

[그림 2] 의미 함수 정의

직관적으로 설명하면 함수 node 는 메시지의 역할 제약 조건(role constraints)을 반환하고 함수 formula는 역할 제약 조건 간의 순서 제약 조건(ordering constraints)을 표현한다. 의미 함수를 좀 더 자세히 살펴보면 함수 sem이 만들어 내는 HDTL 논리식이 다음과 같은 구조를 가지고 있다는 것을 쉽게 보일 수 있다.

$\varphi := \Diamond v.(\phi \wedge \varphi)$
$\quad \mid \Diamond v.\phi$

앞서 예로든 그림 1의 sequence diagram을 의미 함수 sem에 대응시키면 그림 3과 같은 HDTL 논리식을 구할 수 있다.

$\Diamond v_1.(label(v_1) = request$
$\wedge \circledcirc \Diamond v_2.(label(v_2) = acknowledge$
$\wedge snd(v_2) = rcv(v_1) \wedge rcv(v_2) = snd(v_1)$
$\wedge \Diamond v_3.(label(v_3) = release \wedge snd(v_3) = snd(v_1)$
$\wedge rcv(v3) = rcv(v1)$
$\wedge \circledcirc \Diamond v_4.(label(v4) = \infty))))$

[그림 3] 그림 1로부터 생성된 HDTL 논리식

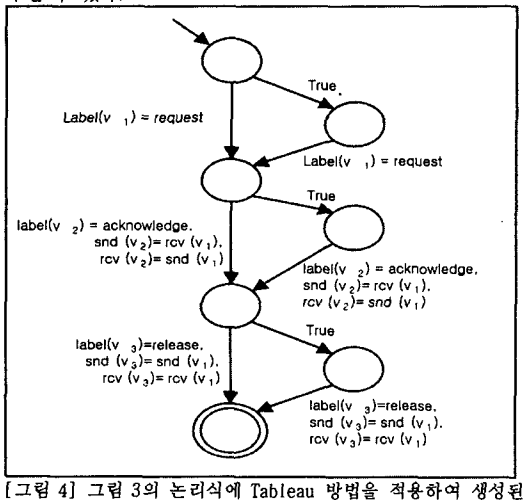
4.2 Tableau Method

HTDL 논리식에서 오라클을 생성하는 데는 기존 시제 논리식을 오토마톤으로 변환하는 Tableau 방법을 변형하여 사용하였다. Tableau 방법의 기본적인 아이디어는 시제 논리식을 현재 상태의 비시제 논리식과 나머지 상태의 시제 논리식으로 나누어 생각하는 것이다. 예를 들어 논리식 □ f는 현재 상태의 비시제 논리식 f와 나머지 상태의 시제 논리식 0 □ f로 나뉘어질 수 있다.

[r∧]	$(f_1 \wedge f_2)\varepsilon \rightarrow \{f_1\varepsilon, f_2\varepsilon\}$
[r∨]	$(f_1 \vee f_2)\varepsilon \rightarrow \{f_1\varepsilon\}, \{f_2\varepsilon\}$
[r⇒]	$(f_1 \Rightarrow f_2)\varepsilon \rightarrow \{(\neg f_1)\varepsilon\}, \{f_2\varepsilon\}$
[r□]	$(\Box f)\varepsilon \rightarrow \{f\varepsilon, (0 \Box f)\varepsilon\}$
[r◇]	$(\Diamond f)\varepsilon \rightarrow \{f\varepsilon\}, \{(\circledcirc \Diamond f)\varepsilon\}$
[rfrz]	$(x.f)\varepsilon \rightarrow \{f\varepsilon[x := \sigma_0]\}$

[테이블 1] 오토마톤 생성에 사용되는 분리 규칙

위의 규칙을 그림 3의 시제 논리식에 적용하면 아래와 같은 오토마톤을 구할 수 있다.



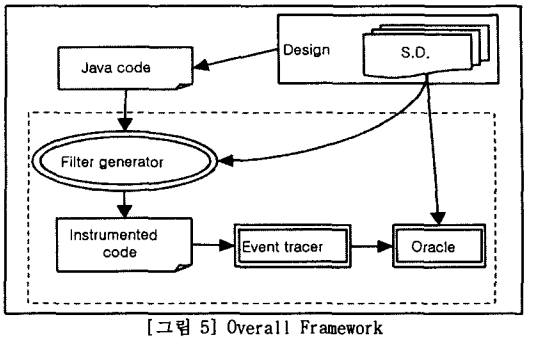
[그림 4] 그림 3의 논리식에 Tableau 방법을 적용하여 생성된 오토마톤

5. Oracle Generation Framework

앞서서 sequence diagram 과 event trace 를 정형적으로 정의하고 sequence diagram에 나타난 정보를 오라클 생성에 사용하기 위하여 HDTL과 의미 함수를 정의하였다. 생성된 HDTL 논리식으로부터 변형된 Tableau 방법을 사용하여 오토마톤을 생성할 수 있다. 생성된 오토마톤은 시스템 내의 객체들이 주고 받는 메시지들의 선후관계에 대한 오라클로 사용할 수 있다. 이 오라클 생성 과정을 그림으로 나타낸 것이 그림 5이다.

Filter generator는 sequence diagram에 표현된 정보를 시스템으로부터 뽑아내기 위해 소스 코드에 탐침(probe)을 삽입하는 모듈이다. Event tracer는 시스템이 수행되면서 여러 객체들이 발생시키는 이벤트들을 수집하여 오라클에 전달해 주는 모듈이다. 오라클은 Tableau 방법을 적용하여 생성된 오토마톤으로 수집된 event trace 가 sequence diagram에 부합되는지 판단한다.

이렇게 생성된 오라클을 다른 테스팅 프레임웍에 적용하여 사용할 경우 기존 테스팅 방법에서 명쾌하게 다루지 못했던 테스팅 결과의 검증 문제를 자동화할 수 있다.



[그림 5] Overall Framework

6. 결론 및 향후 연구과제

객체 지향 시스템의 검증은 기존 절차 지향적 시스템에서 볼 수 없었던 여러 문제들을 발생시켰다. 이 문제들은 주로 객체들의 과거 의존적인(history dependent) 행위나 동적인 배치(dynamic configuration)와 관련이 있다. 우리는 새로운 시제 논리인 HDTL을 도입해서 이 문제를 해결하고자 하였다.

HDTL은 동결 한정사를 사용하여 동적인 시스템의 특성을 표현하고자 하였다. 생성된 HDTL 논리식을 변형된 Tableau 방법을 사용하여 오토마톤을 생성하는 방법을 제안하였고 HDTL 에 기반을 둔 sequence diagram의 의미를 정의하였다. 잘 정의된 Tableau 방법은 시스템의 행위에 이상이 있는지 판단하는 데 사용될 수 있으며 생성된 오토마톤은 객체들이 주고 받는 메시지의 순서 관계에 대한 오라클로서 사용될 수 있다.

이 연구가 정형 분석 기법으로 객체 지향 시스템을 설계하고 테스팅하는 데 도움을 주리라 생각하며 다음과 같이 확장할 수 있다. 먼저 설명한 과정들과 프레임웍을 자동화하는 효과적인 툴의 구현을 생각할 수 있다. 테스팅은 많은 데이터를 반복해서 수행시키는 과정이 필요하다. 이런 여러 번의 수행에서 발생한 결과를 수작업으로 검증하는 것은 무리가 있으며 실수를 발생시킬 가능성이 높다.

그 다음으로 sequence diagram의 의미를 재정의 하는 것이 필요하다. 지금까지 설명한 방법은 에러를 검출하는 방법이 아니다. 단지 sequence diagram 에 부합되지 않는 이벤트의 발생, 즉 이상 상태(anomaly)만을 검출할 수 있다. Sequence diagram의 의미를 강화한다면 이상 행동을 에러로 생각할 수 있을 것이다.

7. 참고 문헌

[1] D. Harel and E. Gery, " Executable Object Modeling with Statecharts," *IEEE Computers*, July 1997.

[2] O. Biberstein, D. Buchs, and N.Guelfi, " Object oriented nets with algebraic specifications: The CO -OPN/2 formalism, " In *Advances in Petri Nets on Object -Orientation*, Springer-Verlag, 1997.

[3] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[4] Laura K. Dillon and Y. S. Ramakrishna, " Generating Oracles from Your Favorite Temporal Logic Specifications," *Proc. 4th ACM SIGSOFT Symp. Foundations of Software Engineering*, San Francisco, pp. 106-117, October 1996.

[5] Laura K. Dillon and Qing Yu, " Oracles for checking Temporal Properties of Concurrent Systems, " *Proc. 2nd ACM SIGSOFT Symp. Foundations of Software Engineering* , New Orleans, pp. 140 -153, December 1994.

[6] Thomas A. Henzinger, " Half-order Model Logic: How to Prove Real-time Properties, " *Proc. Of the 9 th annual symposium on Principles of Distributed Computing*, 1990.

[7] I.jacobson, M. C. Christerson, P. Jonsson, and G.Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1997.

[8] Dennis K. Perters, David Lorge Parnas, " Using Test Oracles Generated from Program Documentation, " *IEEE Transaction on Software Engineering*, vol. 24, no. 3, pp. 161-173, March 1998.