UML 을 이용한 One-Time Password 기술 분석

김영미[0*] 최진영[*] 서동수[**] 김우곤[***]
고 려 대 학 교 컴 퓨 터 학 과[*]
성 신 여 자 대 학 교 전 산 학 과[**]
한 국 정 보 보 호 센 터[***]
{ymkim, choi}@formal.korea.ac.kr

Analysis of One-Time Password Technique using UML

Young-Mi Kim[0*] Jin-Young Choi[*] Dong-Su Seo[**] U-Gon Kim[***]
Dept. of Computer Science & Engineering, Korea University[*]
Dept. of Computer Science, Sungshin Women's University[**]
Korea Information Security Agency[***]

요 약

OMG 에 의해 표준화되어 객체지향 방법론으로 널리 쓰이고 있는 UML 을 이용하여 보안 기술중의 하나인 일회용 패스워드(One-Time Password) 기술을 모델링한다. UML 은 전체적인 시스템을 이해하는데 도움을 준다. 그러나 그래픽컬한 UML 모델링 기술만으로는 불가능한 일관성 및 refinement 체크를 위해 각 다이어그램 특성에 맞게 정형명세나 정형검증을 도구를 적용하는 것이 필요하다. 클래스 다이어그램의 클래스와 정형명세 언어인 Z 스키마(schema)의 유사성을 이용하여 모델링의 정확성을 확인하는데 Z 를 이용할 수 있다.

1. 서론

소프트웨어 시스템의 모델이 점차 복잡하고 방대해지고 있다. 인간의 능력에는 한계가 있어 단순한 의사 소통이나 문서작업만으로 시스템을 완벽하게 이해하는 것이 불가능한 상황이다. 이러한 소프트웨어 시스템은 개발자와 사용자 모두에게 혼란을 가중시키고 있다.

현재 소프트웨어 위기에 대한 해결책으로 객체 개념이 발표되면서 객체 지향 개념을 이용한 프로그램과 방법론이 다양하게 제시되었다. 객체 지향 개발 방법은 소프트웨어 공학에서 추구하는 신뢰성 향상, 유지보수의 용이, 성능의 향상 등의 많은 이점들을 제공한다. 또한 객체 지향 방법론은 실세계의 현상들을 보다 정확하게 모델링 할 수 있기 때문에 개발의 용이성과 분석, 설계, 재사용과 확장성이 좋다. 객체지향 방법론으로 UML, OMT, OOSE 같은 많은 종류의 방법론이 개발되었지만 최근에는 OMG 표준인 UML 이 널리 쓰이고 있다.

그러나 그래픽컬한 UML 모델링 기술만으로는 불가능한 일관성 및 refinement 체크를 위해 각 다이어그램 특성에 맞게 정형명세나 정형검증 도구를 적용하는 것이 필요하다. 설계의 검증이 완전하지 않은 상태에서 시스템을 구현하게 되면 시스템의 오류로 인하여 문제가 발생할 가능성이 크기 때문이다. 에러를 발견하고 나서 수정하는 것이 어렵고 비용 또한 많이 들기 때문에 시스템을 구현하기 전에 어떤 설계의 정확성을 확인하는 것은 중요하다[8].

본 논문에서는 보안기술 중의 하나인 일회용 패스워드 기술을 UML 을 이용하여 모델링하고 모델링된 클래스 다이어그램을 정형명세 언어인 Z 를 적용하여 살펴본다. 2, 3 장에서 UML 과 Z 를 각각 소개하고 4 장은 일회용 패스워드 기술을 UML 을 사용하여 Use-case 다이어그램과 클래스 다이어그램으로 모델링한 것을 보여준다. 5 장에서는 클래스 다이어그램과 정형명세 언어인 Z 와의 관계를 살펴본다.

2. UML 소개

UML 은 기존 객체지향 개발 방법론의 산출물간에 서로 다른 표기법으로 개발자들의 원활한 의사소통에 지장을 주어 객체지향 방법론의 장점이라 할 수 있는 재사용을 어렵게 하기 때문에 새로운 표준으로 제시되었다. UML 기법은 객체지향 모델링이 가져야 할 필수 기능으로 일반화된 표기법을 지원하거나 모델링 요소간의 관계를 명시해주는 등의 기능을 잘 갖추고 있다[4].

UML 의 각 다이어그램은 모델링 요소, 관계, 확장성있는 메커니즘으로 모델링된다. 모델링요소로는 클래스, 인터페이스, 콜라보레이션, Use case, 컴포넌트, 노드등 구조적요소, 인터렉션, 상태기계의 행위적 요소, 패키지, 서브시스템의 집단화 요소, note 와 같은 기타 요소로 구성된다[1]. 관계에는 의존성(dependency), 연관성(association), 일반화(generalization), 구체화(realization)가 있다. 그리고 의미나 문법적으로의 확장을 위해서 제공하는 메커니즘으로 스테레오 타입, tagged 값, 제약(constraint)이 있다. Class 다이어그램은 클래스집합, 인터

페이스, 콜라보레이션, 관계(relationship)을 보여준다. Object 다이어그램은 객체의 집합과 그 관계를 보여준다. Use case 다이어그램은 use case 와 액터(actor)의 집합과 그 관계를 보여준다. 시스템의 행동을 구성하고 설계하는데 특히 중요하다. Sequence 다이어그램과 Collaboration 다이어그램은 일종의 상호작용(객체의 집합과 그 관계로 구성) 다이어그램이다. Sequence 다이어그램은 시간의 순서가 있는 메시지를 강조하고 collaboration 은 메시지를 주고 받는 객체의 구조적인 조직을 강조한다. Statechart 다이어그램 상태기계(state machine)를 보여준다. 실시간 시스템을 모델링하는데 유용하다. Activity 다이어그램은 상태 챠트의 특별한 종류로 액티비티에서 액티비티로의 흐름을 보여준다. 시스템의 기능을 모델링하는데 특히 중요하고 객체간의 제어의 흐름을 강조한다. Component 다이어그램은 컴포넌트 집합간의 구성과 의존성을 보여준다. Deployment 다이어그램은 하드웨어, 소프트웨어를 포함하면서 수행시 처리하는 것의 구성을 보여준다.

UML 은 시스템을 바라보는 다양한 측면과 관점에서 다이어그램을 제시한다. UML 은 세 가지의 측면을 가지고 있다. 기능적인 측면과 정적인 측면(상태) 그리고 동적인 측면(행위)이 있는데 기능적인 측면에는 Use case 다이어그램, Activity 다이어그램이 있고, 정적인 측면에는 Class 다이어그램, Package 다이어그램, Deployment 다이어그램이 있으며 동적인 측면에는 Sequence 다이어그램, Collaboration 다이어그램, State 다이어그램이 있다[2][3]. UML 은 관점에 따라 모델링 단계에서 생성되는 각종 요소를 적절한 표기법과 절차로 도식화함으로써 개발자가 시스템의 전체적인 모습을 정확하게 이해할 수 있도록하는 통찰력을 제공한다.

3. Z 소개

Z 는 1970 년대 후반에서 1980 년대 초반에 걸쳐서 영국의 옥스퍼드 대학(Oxford University)의 프로그래밍 연구그룹(Programming Research Group)의 Jean-Raymond Abrial, Bernard Sufrin 과 Ib SØrensen 에 의해서 개발되었다. Z 언어는 개발 초기서부터 학술적인 범위를 벗어나서 실 시스템 명세에 사용되었다.

Z 는 정형명세 언어이다. Z 언어는 일차 논리(First-Order Logic)와 집합론(Set Theory)과 같은 수학적인 기반을 가지고 있고, 이로 인해서 명세에 많은 이득을 가지고 있다. 예를 들면, 이러한 수학적인 표현은 간결하고, 애매모호함이 없기 때문에 정확한 명세를 할 수 있다. Z 의 기법이 고려하는 시스템의 특성에는 크게 행위에 관한 특성과 자료가 갖는 특성이 있다. 행위에 관한 특성은 자료가 갖는 상태 변화, 즉 입력 자료에 대해 출력자료가 어떤 모습으로 변화하는지에 관한 상태모형을 통해 흔히 표현한다.

어떤 타입의 상수들을 정의하기 위해서 Z 에서 공리기술(Axiomatic description)을 사용한다. 공리기술은 변수를 선언하는 선언부분(a declaration part)과 그 변수의 값에 대한 제약을 나타내는 술부(a predicate part)로 나누어진다. 하지만, 스키마는 이름을 가지고 있다는 점이 공리 기술과 다른 점이다. 따라서, 우리는 이름을 인용함으로써 이 스키마의 내용들을 적용한다. 그리고 스키마 박스를 닫는다. 스키마 안에서 정의된 변수들은 지역적(local)으로 사용된다. 이 변수들은 자신들이 선언된 스키마 안에서만 사용된다. 스키마(schema)는 Z 의 특성을 나타낸다. 스키마 박스는 여러 다른 표현으로부터 Z 명세를 구별하게 해준다. 상태 스키마에서 사용되는 모든 표기들은 일반적인 수학에서 쓰이는 의미와 같다. 스키마는 computing system 을 모델하기 위해서 필요한 새로운 것(저장소인 메모리)을 더한다. 시스템의 메모리의 내용물들은 시스템의 상태라고 불린다. 스키마들은 상태 변수들과 그들의 값의 모임으로 상태를 모형화 한다. Z 상태 스키마 안의 술부는 항상 참이기 때문에 invariant 라고 불린다. 이것은 이 특성은 항상 만족한다는 것을 의미한다. 상태 변수들은 구성요소(component)라고도 불린다. 모든 시스템은 시작을 나타내는 특별한 상태를 가진다. Z 에서는 이러한 상태를 스키마의 이름에 *Init* 을 주는 것으로 나타낸다. 스키마는 다른 프로그래밍 언어에서 제공되는 매크로와 유사하다. 따라서, 스키마 안의 모든 내용을 반복해서 쓰는 것을 대신해서 스키마 이름만을 쓰면 된다. 그 결과 작성자의 시간과 노력을 줄이고, Z 의 명세를 보다 짧게 할 수 있다. 무엇보다도 중요한 점은 작성된 모델의 구조를 독자가 받아들이기 쉽게 해준다. 지금까지 저장소(storage) 상태의 기본적 특성들 중의 하나를 모델링할 때 사용되는 것을 보았다. 변화라는 상태의 다른 면을 모델링 할 때, 즉 사용자가 수정을 함으로써 내용이 변화경우, 이런 종류의 행동을 모델하기 위해서 Z 는 동작 스키마를 제공한다. 종합적으로 총 동작을 스키마로 정의된 개별적인 작은 조각으로 정의하고 스키마 칼큘러스(Schema calculus)를 이용해서 하나로 만든다. 이 방법은 Z 에서 완전한 동작을 정의하는 일반적인 방법이다.

4. 일회용 패스워드 기술 분석

사용자의 ID 와 패스워드를 인증 기반으로 하고 있는 현재의 UNIX 시스템에서 패스워드의 누출은 많은 위험성을 내포하고 있다. 패스워드 도용을 이용한 불법 접속 시도 등 각종 위협에서 전산망을 안전하게 운용하기 위하여 사용자의 패스워드 누출 및 도용 방지, 사용자 신분 위장 및 불법 접속 시동 방지, 그리고 PC 통신망, 금융망 등에 적용 가능한 전산망 원격 사용자 인증 기술의 확보가 필요하다[5]. 4.1 에서 이러한 기술중의 하나인 일회용 패스워드 기술을 설명하고 4.2 에서 UML 을 이용하여 분석한다.

4.1 일회용 패스워드 기술

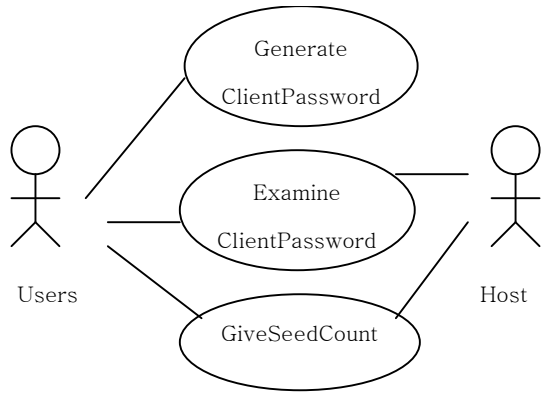
일회용 패스워드 기술은 단방향(one-way) 해쉬 함수를 일정한 수만큼 적용하여 패스워드를 생성한다. 생성된 패스워드는 오로지 한번만 사용되는 일회성을 갖는다[5] 즉, 첫 번째 일회용 패스워드는 사용자의 비밀 패스워드(s)를 정해진 특정 수(n)만큼의 단 방향 함수를 수행하여 생성되어진다. 예를 들어, n=4 라고 가정하면, 첫 번째 일회용 패스워드 p(1) = f(f(f(f(s))))를 가진다. 다음 일회용 패스워드는 사용자의 비밀 패스워드(s)를 단 방향 함수에 n-1 번 적용하여 생성되어진다. 그러므로 다음 일회용 패스워드는 p(2) = f(f(f(s)))를 갖게 된다.

이렇게 생성되는 일회용 패스워드는 p(i)의 사용을 모니터하고 있는 도청자가 다음 패스워드 p(i+1)를 생성해 낼 수 없게 함으로써 패스워드의 누출 위험성을 줄일 수 있다. 처음 시점에서 사용자의 비밀키를 알지 못하면

도청이 불가능하게 되기 때문이다.
그러나 사용자에 의해 수행되는 단 방향 함수의 수가 하나씩 줄어들기 때문에, 어느 시점에 다다르면 사용자는 시스템을 재 초기화 해야만 한다.
사용자가 전송한 일회용 패스워드의 검사를 하기 위해 처음에 호스트 컴퓨터는 수신한 일회용 패스워드의 복사 본을 저장하고, 그것을 단 방향 함수에 적용한다. 만약 그 결과가 시스템의 패스워드 파일 안에 저장된 복사본과 일치하지 않으면, 그 인증 요구는 실패하게 된다. 만약 그들이 일치하면, 시스템 패스워드 파일 안에 있는 사용자의 엔트리는 단 방향 함수의 마지막 실행 전에 저장되어 있던 일회용 패스워드의 복사 본으로 갱신되어진다.

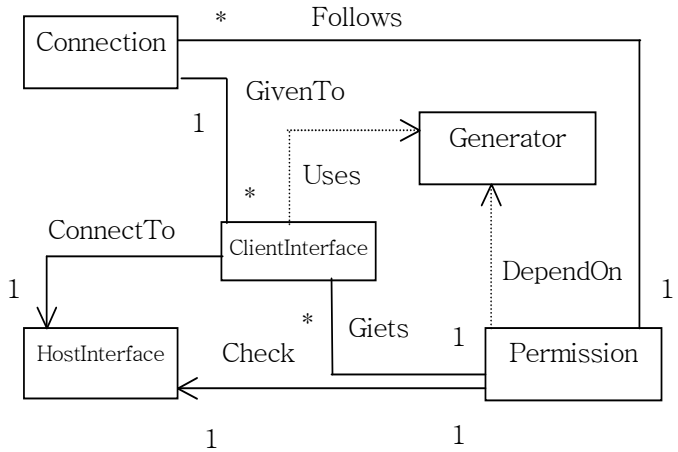
4.2 UML 을 이용한 일회용 패스워드 기술 분석

다음은 UML 을 이용하여 4.1 에서 설명된 일회용 패스워드 기술을 정확하게 반영되도록 Use Case 다이어그램과 클래스 다이어그램으로 모델링한 것이다.



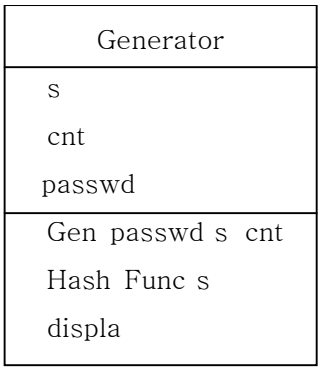
(그림 1) 일회용 패스워드에 관한 Use Case 다이어그램

(그림 1) 에서 보여진 것처럼 일회용 패스워드는 크게 생성, 검사, 부여의 기능을 갖는 세 가지 Use Case 로 나눌 수 있다. Host 가 User 에게 seed 와 count 를 제공하고 사용자는 부여받은 seed 와 count 를 이용하여 일회용 패스워드를 생성한다. 이렇게 생성된 패스워드는 Host 에 의해 검사되어 인증결정을 한다.



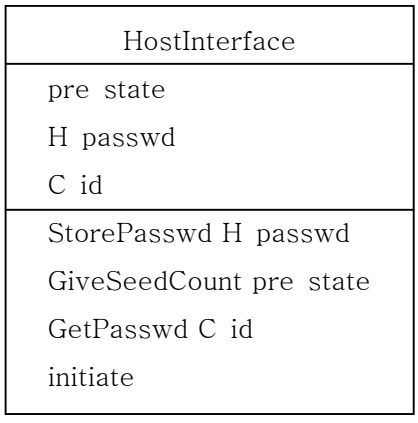
(그림 2) One-Time Password Class Diagram

(그림 2) 는 One-Time Password 시스템을 구성하기 위한 클래스와 그것들의 관계를 보여준다. Generator 클래스는 ClientInterface 와 Permission 클래스에 dependency 관계이고 그 외의 것들은 association 관계이다. ClientInterface 클래스와 HostInterface 클래스 관계에서 ClientInterface 클래스는 HostInterface 클래스쪽으로 navigability 를 가리킨다. HostInterface 클래스는 클라이언트에게 일회용 패스워드를 생성할 수 있도록 하는 seed 와 count 를 준다. Permission 클래스는 사용자가 로그인한 id 가 올바른 것인지를 확인하고 그 id 에 맞는 패스워드인지 검사한다. ClientInterface 클래스는 사용자의 id, 패스워드, seed, count 등의 값을 입력하도록 요구하여 각각의 특성에 맞게 처리한다. Connection 클래스는 사용자와 호스트의 연결관리를 해준다.



(그림 3) Generator 클래스

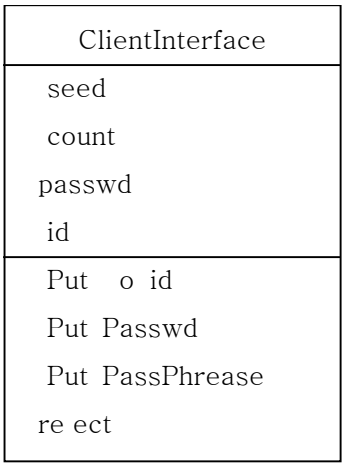
(그림 2)의 Generator 클래스를 좀더 구체적으로 살펴보면 (그림 3)과 같다. Generator 클래스는 사용자의 seed 를 정해진 특정 수만큼의 해쉬 함수를 수행하여 일회용 패스워드를 생성하여 사용자에게 보여준다. Attribute s 는 사용자가 입력한 비밀 패스워드 값을 가지며, cnt 만큼 해쉬함수를 적용하여 일회용 패스워드를 계산하여 display method 를 호출하여 그 값을 사용자에게 보여준다. Hash_func 는 s 를 parameter 로 가지며 패스워드를 생성하여 생성된 값을 return 한다. Display 는 gen_passwd 에서 계산된 비밀패스워드를 사용자가 사용할 수 있도록 사용자에게 보여준다.



(그림 4) HostInterface 클래스

HostInterface 클래스는 클라이언트에게 일회용 패스워드를 생성할 수 있도록 하는 seed 와 count 를 주고 사용자가 호스트에 접속할 때 쓰이는 패스워드를 저장한다. 재

로운 seed 와 count 를 줄 수 있도록 하는 initiate 라는 초기화 method 를 갖는다. H_passwd 는 사용자의 패스워드 값을 가지며 StorePasswd method 에서 호스트에 값을 저장하기 위해 사용된다. C_id 는 사용자의 id 에 맞는 패스워드를 호스트에서 가져오는 데 이용된다. pre_state 는 사용자가 호스트로부터 새로운 seed 와 count 를 부여받을 수 있도록 하는 boolean 값이다. 만약 pre_state 가 false 이면 새로운 seed 와 count 를 사용자에게 주고 이미 password 를 생성할 수 있는 seed 와 count 를 사용자가 가지고 있을 때는 true 인 상태를 유지한다. GiveSeedCount 는 사용자에게 seed 와 count 를 전송한다. seed 는 난수함수를 이용하여 생성하고 count 값 역시 난수로 값을 결정하되 해쉬함수 적용횟수를 고려하여 난수 범위를 한정한다. 이렇게 생성된 seed 와 count 값을 사용자에게 전송하고 이때, pre_state 값을 false 인 상태에서 true 값으로 바꾼다. 또한 생성된 seed 를 count-1 만큼 해쉬함수에 적용하여 호스트의 패스워드 값에 저장한다. StorePasswd 는 호스트가 사용자로부터 수신한 일회용 패스워드를 저장한다. 이것은 저장된 패스워드와 사용자가 다음에 접속하였을 때의 패스워드를 비교하여 인증 요구의 허락여부를 결정하기 위함이다. GetPasswd 는 사용자의 id 를 입력으로 받아 호스트 데이터베이스 서버에서 패스워드를 가져와서 그 값을 return 한다. initiate 는 해쉬함수에 적용하는 횟수를 나타내는 count 가 0 에 다다를 때 사용자에게 새로운 seed 와 count 를 주기 위해 pre_state 를 false 로 초기화한다.



(그림 5) ClientInterface 클래스

ClientInterface 클래스는 사용자의 id, 패스워드, seed, count 등의 값을 입력하도록 요구하여 각각의 특성에 맞게 처리한다. seed, count 는 일회용 패스워드를 생성하는데 사용된다. passwd 와 id 는 패스워드와 접속하는 사용자의 id 를 저장하기 위함이다. Put_Logid 와 Put_Passwd 는 id 와 Passwd 를 입력하도록 요구하고 입력을 마친 경우 각각 올바른 값인지 확인하기 위해 Permission 클래스에 있는 method 를 호출한다. Put_PassPhrase 는 로컬 시스템에서 일회용 패스워드를 생성할 수 있는 권한을 부여하기 위해 패스워드 입력을 요구하고 올바른 값일 때는 Generator 클래스의 Gen_passwd method 를 호출하여 값을 생성하고 그렇지 않은 경우 ClientInterface 클래스의 reject method 를 호출한다.

지금까지 UML 을 이용하여 일회용 패스워드기술을 모델링해보았다. UML 을 통해 모델링 단계에서 생성되는 각종 요소를 적절한 표기법과 절차로 도식화함으로써 시스템의 전체적인 모습을 정확하게 이해하는데 도움을 주었다. 그러나 그래픽컬한 UML 모델링 기술만으로는 불가능한 일관성 및 refinement 체크를 위해 각 다이어그램 특성에 맞게 정형명세나 정형검증을 도구를 적용하는 것이 필요하다. 4 장에서는 UML 다이어그램중의 하나인 클래스 다이어그램과 정형명세 언어 Z 와의 관계을 살펴본다.

5. 클래스 다이어그램과 정형명세 언어 Z

(그림 3) Generator 클래스의 method Hash_func 은 해쉬 알고리즘인 MD4 나 MD5 로 구현할 수 있다. 해쉬 알고리즘이란, 전자서명에서 송신자 이외에 제 3 자에 의한 문서 위조를 방지하는 부인 방지 서비스를 제공하기 위해서 필수 조건으로 필요하다. 전용 해쉬 알고리즘으로 대표적인 것이 MD4, MD5 알고리즘이다.
MD4 는 보안 표준의 하나로 일종의 해쉬 알고리즘으로 그 목적은 해쉬함수의 목적과 동일하다. 공개키 암호화 기법(public - key cryptosystem)과 함께 쓰여 메시지 인증이나 전자 서명 기법(부가형 전자 서명 기법 : digital signature with appendix)에 사용된다. 입력(input)으로 임의의 길이(arbitrary length)의 메시지(message)를 받아서 출력(output)으로 128 비트의 지문형 message digest 를 생성한다.
Generator 클래스 일부의 method 를 Z[7]를 이용하여 명세할 수 있다. 5.1 에서 해쉬함수 부분에 MD4 로 구현된 것을 Z 를 이용하여 정형명세한 일부분를 보여준다.

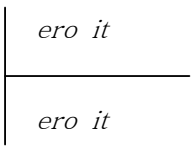
5.1 MD4 를 Z 로 명세

Z 명세는 명세에 사용될 데이터 타입, 집합, 전역 상수를 선언하고, 시스템의 추상화 상태를 정의한다. 그런 후에 시스템의 동작을 성공적인 경우와 에러 조건인 경우를 정의하고, 두 경우를 합쳐서 완전한 동작으로 명세한다.

MD4 알고리즘은 비트 입력에 대한 연산을 수행한다. 비트는 0, 1 로만 이루어진 자료형으로 정의한다.

$$\boldsymbol{BIT} == \{0, 1\}$$

패딩 단계에서 입력된 메시지의 길이를 512 로 나누어서 나머지가 448 이 되도록 모자라는 부분을 '0'로 채운다. 다음은 패딩되는 '0'비트에 대한 정의이다.



(그림 6) 은 패딩단계 명세의 일부이다.
MD4 알고리즘에서는 패딩을 두 단계로 나누어서 진행한다. 첫 단계는 '1'과 '0'비트를 이용하여 448 비트(mod 512)를 만들어주는 단계이고, 두번째는 입력된 메시

길이를 64 비트로 만들어서 패딩된 뒷 부분에 합치는 단계이다. 아래의 Z 코드중 padding1 은 메시지에 '1'비트 하나를 패딩하는 단계이고, zeropadding 은 입력에 일정수의 '0'비트를 패딩하는 부분이고, padding 은 입력을 padding 과 zeropadding 을 이용하여 448 비트(mod 512)를 만드는 부분에 대한 명세이다.

paddin 1 se $IT_{\rightarrow}$ se IT

$\forall$ *messa e* se $IT \bullet$ *paddin* 1 *messa e* *messa e* ^ 1

eropaddin $_{N\rightarrow}$ se IT

$\forall$ *num* N •
eropaddin num
if *num* 1
then *ero it*
else *ero it* ^ *eropaddin num* _ 1

paddin se $IT_{\times}$ $_{\rightarrow}$ se IT

$\forall$ *messa e* se $IT \bullet$
paddin messa e messa e
if *messa e* mod 1
then *paddin* 1 *messa e* ^ *eropaddin* 1
else if *messa e* mod 1
then *paddin* 1 *messa e* ^
eropaddin messa e
else *paddin* 1 *messa e* ^
eropaddin messa e

(그림 6) 패딩 단계 명세의 일부

위와 같이 클래스 메소드를 Z 로 명세할 수 있다. 그러나 위의 명세는 단지 개별의 클래스에 대한 명세이다. 5.1 에서 클래스들의 관계가 나타나는 클래스 다이어그램을 Z 로 변환을 일회용 패스워드 클래스 다이어그램을 통해 살펴본다.

5.2 Z 로 클래스 다이어그램을 정형화

그래픽한 표현은 실제 모델에 대한 정확성을 높이지만 확실한 의미적인 기반이 부족하다. 이것은 오히려 모델에 대한 이해를 어렵게 하므로 기호의 정확한 의미적인 이해가 바탕이 되어야 한다. 또한 시스템을 구현하기 전에 에러를 발견하고 나서 수정하는 것은 어렵고 비용 또한 많이 들기 때문에 어떤 설계의 정확성을 확인하는 것은 중요하다. 그래서 정형기법과 비정형기법을 통합하려는 연구가 활발하게 진행되고 있다. 좀더 대중적인 비정형기법과 정형기법을 통합(integrate)하는 이유는 크게 3 가지로 볼 수 있다. 첫번째는 비정형기법의 어플리케이션을 확장하기 위해서다. 즉 비정형기법은 정형기술의 어플리케이션을 확장할 수 있다. 두번째는 소프트웨어 개발에 좀더 정형적 접근의 사용을 위한 진보적인 단계를 제공한다. 마지막으로 엄격한 단계로 구분되어있는 모델링에서 다양한 단계통해 행위와 구조의 명세를 뒷받침하기 위해서이다[6].

이 장에서는 클래스 다이어그램의 의미적인 이해를 바탕으로 Z 를 이용하여 클래스 다이어그램을 정형화해 보았다. 정형화하는데 사용된 규칙은 FuZe(Fusion/Z Environment) 툴에서 적용하는 것을 기본으로 하였다[6][9]. 클래스 다이어그램에 포함된 정보는 직접 어플리케이션을 구현하기 위해 쓰여질 소스코드로 연결되므로 클래스 다이어그램은 중요한 다이어그램이라 할 수 있다. Z 의 스키마(schema) 구조는 객체구조와 정형적으로 표현된 설명간의 분명한 연결을 제공하면서 클래스의 개념과 클래스 구조로 직접적으로 연결될 수 있다[6]. 클래스와 관련된 불변의 특성(invariant)은 Z 스키마의 서술부에 나타난다. 클래스의 어트리뷰트의 타입 이름은 Z 에서 그대로 타입 이름으로 이어진다. 이것은 기본형(basic types)이나 스키마로 정의해야 한다. 기본형은 원시적 요소들의 집합이다. 만약 타입이 없다면 대문자로 된 이름이 사용되고 Z 의 기본 타입으로 선언된다

(그림 2)의 일회용 패스워스를 클래스 다이어그램으로 모델링한 것 중 HostInterface 클래스와 ClientInterface 클래스를 Z 로 변환하여 보겠다.

HostInterface 클래스의 정의는 다음과 같다.

기본형

[Host_ID, PRE_STATE, H_PASSWD, C_ID]

HostInterface 클래스 스키마

HostInterface

ident Host ID O ect Identifier
Pre state P E ST TE
H passwd H P SS D
C id C ID

각 클래스의 어트리뷰트는 위와 같이 기본형으로 선언이 된 후 스키마에서 변수에 대한 타입으로 사용된다.

ClientInterface 클래스도 HostInterface 클래스와 유사하게 정의하면 다음과 같다.

기본형

[Client_ID, SEED, COUNT, PASSWD, ID]

ClientInterface 클래스 스키마

ClientInterface

ident Client ID O ect Identifier
seed SEED
count COU T
passwd P SS D
id ID

위의 정의에서 클래스의 메소드부분은 정의하지 않았지만 각각의 메소드를 5.1 장에서 MD4 를 Z 로 명세한 것과 같은 형식으로 다루면 될 것이다.
HostInterface 클래스와 ClientInterface 클래스는 연관성(association)으로 연결되어있다.

다음은 연관성을 Z 스키마로 변환한 것이다

ConnectTo

host $\mathbb{P}$ Host
client $\mathbb{P}$ Client
connect to Host $\leftrightarrow$ Client

---

connect to $\in 1 \leftrightarrow *$ Host Client

HostInterface 클래스와 ClientInterface 클래스간의 연관에서 HostInterface 클래스 쪽으로 navigablity 를 갖는데 이것은 Z 로 나타내지 않았다. 선 아래부분의 서술부는 Z 매크로라고 부르는 Z 에 대한 정의의 확장의 예이다. Ont-to-many 관계로서 connect_to 관계를 정의하는 부분이 서술부에 나타나있다.
지금까지 HostIterface 클래스와 ClientInterface 클래스의 관계를 Z 로 변환해 보았다. 하지만 각 클래스간의 관계는 여기서 다루지 않은 의존성, 일반화, 구체화 등, 연관성 이외에 것은 아직 그 부분까지는 다루지 못했다. 그리고 본 논문은 FuZe 에서 제공하는 규칙에 따라 일회용 패스워드라는 예를 가지고 Z 로 변환한 것이다.
이렇게 클래스 다이어그램에 나타나는 정보를 잃지 않고 그대로 정형명세 언어인 Z 로 변화함으로써 모델링의 정확성을 확인할 수 있다. 그러나 클래스 전체에 대한 Z 명세와 클래스간의 관계까지를 완벽하게 정형명세 해야 할 필요가 있다.

5.결론

UML 은 객체지향 방법에 대한 전문 개발자들에 의해 개발된 현재의 객체지향 모델링을 위해 제시된 표준 언어이다. 객체 지향 모델링 구조의 단순하고 그래픽컬한 특성은 분명하고 시각적인 모델의 생성을 쉽게 하지만 정확성을 검증하기에 부족한 점이 있다. 따라서 UML 의 각 다이어그램의 특성에 맞는 정형기법을 도입하여 모델링에 정확성을 확인할 필요가 있다. 클래스는 정형명세 언어인 Z 스키마의 표현적인 유사성이 있다.
이것은 클래스 다이어그램 모델링의 정확성을 확인하는데 Z 를 사용할 수 있도록 한다. 본 논문에서 클래스 다이어그램의 극히 일부분을 Z 로 정형화시켜 보았다. 그러나 클래스 메소드부분이나 어트리뷰트를 정의하는 수준에 지나지 않다.
앞으로 클래스를 연결시키는 관계를 모두 정형화할 수 있도록 클래스 다이어그램의 의미적인 면을 명확히 하여 Z 의 관계 정확하게 변형할 수 있어야 한다. 이렇게 함으로써 클래스 다이어그램 자체를 완벽하게 정형명세하여 보다 정확한 모델링 결과를 얻을 수 있다.

6. 참고 문헌

[1] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, Addison Wesley, 1999.
[2] Martin Fowler, UML Distilled, Addison Wesley, 1999.
[3] Alhir, UML IN A NUTSHELL, O'REILLY, 1998.
[4] UML Notation Guide, Rational Software, 1997.
[5] http://www.kisa.or.kr/techonology/sub4/password.htm .
[6] Malcolm Shroff & Robert B. France, Towards a Formalization of UML Class Structures in Z
[7] Andrew Harry Formal Methods Fact File, VDM and Z, John Wiley & Sons
[8] M.Hinchey, J. Bown, Application of Formal Methods, Prentice-Hall, 1995
[9] http://www.cse.fau.edu/research/MIRG
[10] J. M. Spivey. The Z Notation: A Reference Mannual. Prentice Hall, Englewood Cliffs, NJ, Second edition, 1992.