韓國컴퓨터情報學會 論文誌
第15卷 第6號, 2010. 6.

2010-15-6-1-1

# SMS 부호화 복호화 모듈 검증 방법에 대한 연구

최 광 훈*

# A Study on the Verification Scheme of SMS Encoding and Decoding Module

Kwanghoon Choi *

## 요 약

본 논문에서는 3GPP(3rd Generation Partnership Project)에서 정의한 SMS PDU (Protocol Data Unit) 포맷을 주어진 SMS 부호화 복호화 모듈에서 정확하게 구현했는지 검증하는 방법을 제안한다. 기존 SMS 관련 도구들은 SMS 게이트웨이를 통해 송수신하거나 또는 SMS PDU 해석을 목적으로 개발되어 3GPP에서 정의한 세부 SMS PDU 규격에 따라 정확히 구현했는지 테스트하는 용도로는 적합하지 않다. 본 논문에서 제안한 방법은 함수형 언어 Haskell로 작성된 QuickCheck 라이브러리를 활용해 3GPP에서 정의한 구조에 맞는 SMS PDU 테스트 데이터를 자동 생성하여 SMS 부호화 복호화 모듈을 테스트한다. C언어로 작성된 리눅스 모바일 플랫폼 SMS 모듈에 적용하여 이 모듈의 부호화 복호화 기능을 테스트한 결과 BCD 포맷 시간 정보를 잘못 해석하는 사례 등 중요한 오류들을 발견할 수 있었다. 제안한 방법은 3GPP에서 정의한 규격에 맞추어 SMS PDU를 생성하기 때문에 일반적인 SMS 모듈들에 모두 적용 가능한 장점을 지닌다. 본 논문에서 사용한 방법과 같이 QuickCheck 라이브러리를 통해 다른 네트워크 프로토콜 데이터 규격에 대한 부호화 복호화 검증에도 응용할 수 있을 것이다.

## Abstract

This paper proposes a test method for compliance of SMS encoder and decoder modules with 3GPP (3rd Generation Partnership Project) specification on SMS PDU (Protocol Data Unit). The existing tools have focused on providing an SMS gateway and on helping to view and edit a single SMS PDU, which rarely help to resolve the compliance test problem. The proposed compliance test method is based on an automatic generation of SMS PDUs fully compliant with the 3GPP specification by using QuickCheck library written in Haskell. By applying the proposed method to a C-based SMS encoder and decoder in Linux Mobile platform, we have found out several critical bugs such as wrong interpretation of time stamps in BCD format. The automatic SMS PDU generator is reusable in that it only depends on the 3GPP SMS specification. The QuickCheck library is also applicable for testing other network protocol data encoders and decoders, as used in this paper.

▸ Keyword : 테스트 (Testing), SMS, 자동 테스트 데이터 생성 (Automatic Test Data Generation), Haskell, QuickCheck

• 제1저자 : 최광훈
• 투고일 : 2010. 03. 30, 심사일 : 2010. 04. 06, 게재확정일 : 2010. 05. 15.
* LG전자 책임 연구원

# Ⅰ. 서 론

소프트웨어의 복잡도가 크게 증가함에 따라 소프트웨어 오류 종류가 다양해지고 빈도 또한 높아진다. 휴대폰 모바일 소프트웨어의 경우도 예외일 수 없는데 2010년 1월1일에 발생한 국내 휴대폰 단말기 SMS 문자 연도 표시 오류도 그러한 사례이다[1]. SMS 메시지를 받았을 때 수신 연도가 2010년이 아닌 2016년으로 표기되는 현상이 발생했는데 이는 SMS 메시지에서 BCD (Binary-Coded Decimal) 포맷의 수신 연도 값 0x10을 16으로 잘못 해석하여 발생했다. 이러한 사소한 소프트웨어 오류로 총 73종 휴대폰의 소프트웨어를 업그레이드 했고 대외적인 브랜드 이미지 추락 등 유무형의 비용이 들었다.

소프트웨어 오류를 찾는 방법은 크게 테스트(Testing), 인스펙션(Inspection), 정적 분석(Static Analysis)이 있다. 본 논문은 앞서 소개한 SMS 연도 표시 오류와 같이 SMS 메시지 해석기의 오류를 찾는 테스트 방법을 주제로 한다. 즉 3GPP (3rd Generation Partnership Project) SMS 메시지 PDU (Protocol Data Unit) 구조의 규격[9,10]에 따라 SMS 부호화 (Encoding) 복호화 (Decoding) 모듈을 정확히 구현했는지 테스트하는 방법을 다룬다.

기존의 SMS 테스트를 위해 사용된 도구는 SMS PDU 뷰어와 SMS 게이트웨이가 있다. SMS PDU 뷰어를 통해서 특정 SMS 메시지를 해석하거나 특정 필드를 변경해서 다시 부호화할 수 있다. 하지만 개발자가 각 필드에 대해 원하는 값을 수동으로 일일이 입력해야하는 제약점이 있다. 많은 SMS 메시지 를 자동으로 생성하는 기능을 제공하지 않는다.

SMS 게이트웨이는 웹을 통해서 작성한 SMS 메시지를 실제 모바일 디바이스가 연결된 모바일 네트워크로 송수신하기 위한 서버이다. SMS 게이트웨이를 통해서 SMS 메시지 송수신 기능을 테스트하기에 용이하지만 SMS 메시지 구조의 세부 필드들의 부호화 복호화 과정을 테스트하는데 사용하기에는 불필요한 과정이 많다. 예를 들어 이 테스트를 위해 굳이 모바일 네트워크 망을 이용할 필요가 없다.

앞서 설명한 바와 같이 SMS 메시지 부호화 복호화 과정을 기존의 SMS 도구들로 테스트하는데 제약점이 있다. 본 논문은 SMS 메시지 부호화 복호화 과정을 테스트하기 위해 기존의 SMS 도구들 보다 더욱 적합한 SMS PDU 데이터 자동 생성 방법을 제안하고 실제 SMS 모듈 테스트에 적용한 사례를 공유하고자 한다.

본 논문에서 함수형 언어 Haskell로 작성된 테스트 데이터 생성 라이브러리 QuickCheck을 활용해서 3GPP SMS PDU 규약의 SMS PDU를 자동으로 만들어 SMS 부호화 복호화 모듈을 테스트하는 방법을 제안한다. QuickCheck 라이브러리는 주어진 Haskell 타입에 대해 그 타입의 값을 무작위로 생성할 수 있는 함수를 정의하도록 도와준다. 즉 3GPP SMS PDU 구조를 Haskell의 타입들로 표현하고, 이 타입들에 대한 랜덤 함수를 QuickCheck 라이브러리를 활용해 정의한 다음, 이 랜덤 함수들로부터 나온 각 필드 값들이 서로 일관성을 갖도록 조합함으로써 궁극적으로 스펙에 맞는 SMS PDU를 생성하는 랜덤 함수를 만들 수 있다. 본 논문에서 구현한 프로그램은 [11]에서 다운로드 받을 수 있다.

본 논문은 다음과 같이 구성되어 있다. Ⅱ장에서 SMS 개요와 SMS 테스트 도구에 대한 관련 연구를 소개하고 테스트 데이터 자동 생성 방법과 QuickCheck 라이브러리에 대해 설명한다. Ⅲ장에서 함수형 언어 Haskell로 3GPP SMS PDU 구조를 표현하는 방법과 QuickCheck 라이브러리를 활용해 SMS PDU를 자동으로 생성하는 방법을 설명하고 SMS 부호화 복호화 모듈 테스트 방법을 제안한다. Linux Mobile 플랫폼의 SMS 모듈에 100개의 SMS PDU를 생성해 적용한 결과를 공유하고, 본 논문에서 제안한 방법에 관한 제약 사항과 극복 방안을 논의한다. Ⅳ장에서 향후 연구 주제를 논의하고 결론을 맺는다.

# Ⅱ. 관련 연구

## 1. SMS (Short Message Service) 개요

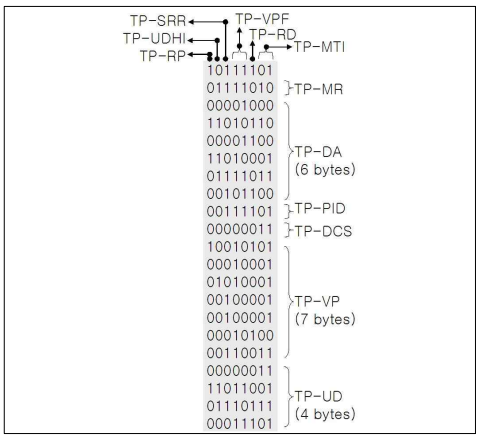


그림 1. 3GPP 규격[9,10] SMS PDU 예
Fig. 1. An SMS PDU in 3GPP Specification[9,10]

SMS는 모바일 네트워크에 연결된 모바일 디바이스들이 제한된 크기의 텍스트 메시지를 서로 주고받을 수 있는 서비스이다 [14]. SMS 모듈은 메시지 송수신 모듈과 메시지 부호화

(Encoding) 복호화 (Decoding) 모듈로 구성되어 있다. SMS의 전형적인 시나리오는 다음과 같다. 사용자가 UI (User Interface)를 통해 문자 메시지를 입력하면 이를 SMS PDU (Protocol Data Unit)로 부호화한다. SMS 프로토콜에 따라 SMSC (Short Message Service Center)에 이 SMS PDU를 보낸다. SMSC에서 수신 전화번호로 이 SMS PDU를 보내면 해당 수신자 모바일 디바이스에서 이를 복호화한다. 사용자에게 UI를 통해 수신된 문자 메시지 내용을 보여준다.

문자 메시지 "You"를 SMS PDU로 부호화하면 <그림 1>과 같다. SMS PDU는 기본적으로 바이트 단위로 구성되어 있다. 3GPP SMS 스펙 [9,10]에서 각 바이트들과 각 비트들이 어떤 역할을 하는지 정의한다.

<그림 1>에서 3GPP 스펙에서 정의하는 SMS PDU의 각 필드들을 보여준다. TP-MTI "01"은 이 SMS PDU가 송신 메시지임을 뜻한다. TP-DCS "00000011"은 GSM 7비트 방식[10]으로 문자 메시지를 부호화함을 나타낸다. TP-UD는 SMS 메시지 "You"를 GSM 7비트 방식으로 표현한 것으로 TP-UD의 첫 번째 바이트 "00000011"은 메시지 길이가 3임을 뜻한다.

## 2. SMS 테스트 도구

### 2.1 SMS PDU 뷰어 (Viewer)

SMS PDU는 16진수 바이트로 표현되어 있으므로 이 데이터를 쉽게 읽을 수 있는 형태로 변환하는 도구가 필요하다. 이와 같은 기능을 제공하는 SMS PDU 뷰어로 PDUSpy (http://www.nobbi.com/pduspy.html)와 ACCESS의 Virtual Phone (http://www.accessdevnet.com/)이 있다.

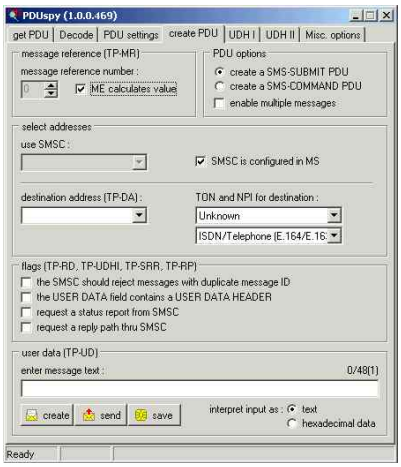


그림 2. SMS PDU 유틸리티 (PDUSpy)
Fig. 2. An SMS PDU Utility, PDUSpy

SMS PDU 뷰어는 보통 편집 기능도 함께 제공해서 SMS PDU의 세부 필드들을 변경할 수 있다.

PDUSpy나 Virtual Phone과 같은 SMS 도구들은 테스트용 SMS PDU를 자동으로 생성하는 기능을 제공하지 않는다. 이 도구로 테스트 SMS PDU를 만들기 위해서 개발자가 각 필드 별로 원하는 값을 수동으로 지정해 주어야 한다. SMS PDU 부호화 복호화 모듈을 테스트하기 위해 다양한 SMS PDU 값을 만드는데 많은 비용이 들 것이다.

### 2.2 SMS 게이트웨이

SMS 게이트웨이는 TCP/IP 네트워크와 모바일 네트워크 사이에 SMS PDU를 서로 보내고 받을 수 있는 서버이다. <그림 3>의 아키텍처로 구성되어 있다. 일반적으로 웹 인터페이스를 통해 문자 메시지와 수신 번호 등을 기술하고 TCP/IP 네트워크를 통해 SMS 게이트웨이에 전달한다. SMS 게이트웨이에서 이를 모바일 네트워크로 전송하면 해당 수신 번호의 휴대폰을 통해 이 문자 메시지를 수신 받는다. 상용 SMS 게이트웨이 솔루션으로 NowSMS (http://www.nowsms.com)가 있다.

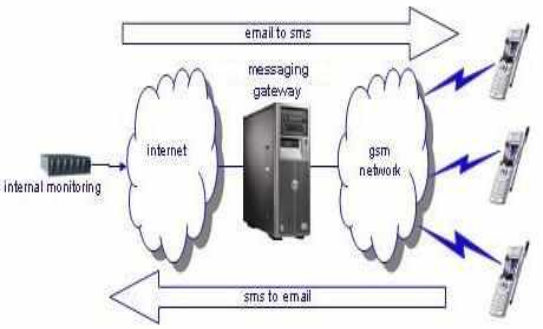


그림 3. SMS 게이트웨이
Fig. 3. SMS Gateway

SMS 게이트웨이를 이용하면 문자 메시지를 휴대폰으로 직접 받을 수 있으므로 SMS 수신 프로토콜을 테스트하기에 쉽다. 하지만 모바일 네트워크 망 환경이 갖추어지지 않았거나 수신 가능한 단말기가 없으면 SMS 게이트웨이를 사용하기 어렵다. 모바일 네트워크 망을 사용해야 하므로 많은 SMS PDU를 송수신하려면 시간도 오래 걸린다. 또한 SMS 게이트웨이 사용자 인터페이스에서 SMS PDU 필드들을 다양하게 조합해서 만들 수 있는 환경을 현재 제공하지 않는다. 이러한 이유 때문에 SMS 메시지 부호화 복호화 모듈을 효율적으로 테스트하기에 적합하지 않다.

### 2.3 SMS 관련 오픈 프로토콜 및 도구

PC와 모바일 디바이스 간 혹은 PC와 PC간 SMS 메시지 전송을 위해 오픈 소스 기반 SMS 도구와 라이브러리가 다수 존재한다. SMS 부호화 복호화 모듈을 테스트하기 위한 환경을 만드는데 이 도구나 라이브러리를 이용할 수 있다.

첫째, 모바일 네트워크 용 모뎀이나 모바일 폰을 PC와 연결하여 PC 프로그램으로부터 SMS 메시지를 송수신하는 방법이다. gnokii (www.gnokii.org), gammu (www.gammu.org), gsmlib (www.pxh.de/fs/gsmlib), kannel (www.kannel.org), SMSLib (smslib.org)가 있다. 참고로 SMSLib는 SMS 메시지를 XML로 표현해 다룬다.

둘째, SMPP (Short Message Peer-to-Peer) 프로토콜 도구이다. SMPP 프로토콜은 PC와 SMSC사이에서 TCP/IP 기반 SMS 메시지 송수신 기능을 제공한다. SMS 포럼 (www.smsforum.net)에서 SMPP 클라이언트 테스트 툴 (SCTT)과 SMPPsim (www.seleniumsoftware.com) 등을 제공한다.

## 3. 테스트 데이터 자동 생성 방법

테스트는 소프트웨어 품질을 보장하는 실용적인 방법이지만 그 비용은 전체 소프트웨어 개발비의 절반에 해당할 정도로 많이 든다. 따라서 대량의 테스트를 쉽게 진행하고 주기적으로 반복 적용할 수 있는 자동화 테스트 방법이 필요하다.

자동화 테스트 방법을 설계할 때 테스트 데이터를 자동으로 생성하는 방법, 테스트 결과를 자동으로 판단하는 방법, 테스트 대상 모듈을 충분히 검증했음을 확인하는 방법을 고려해야 한다.

### 3.1 QuickCheck 라이브러리 기반 테스트 데이터 자동 생성 방법

함수형 언어 Haskell로 작성된 QuickCheck 라이브러리는 테스트 데이터를 자동으로 생성하는 방법을 제공한다. QuickCheck 라이브러리는 Haskell로 정의된 타입에 대해 해당 타입의 값을 무작위로 생성하는 함수를 정의하는 방법을 제공한다. 즉 주어진 Haskell 타입에 대해 이 타입을 구성하는 서브 타입의 값을 무작위로 만드는 함수들을 먼저 정의하고, 주어진 타입과 서브 타입들이 어떻게 구성되어 있는가에 따라 서브 타입들에 대한 랜덤 함수들을 조합해서 궁극적인 랜덤 함수를 정의한다.

Haskell의 타입은 정수 타입과 같은 기본 타입 (primitive type), 불(Bool) 타입과 같은 선택적 타입 (sum type), 둘 이상의 타입으로 구성된 구조화 타입 (product type)으로 정의된다. 각 유형의 타입들에 대한 랜덤 함수를 QuickCheck 라이브러리를 활용해서 만드는 방법을 설명한다.

먼저 정수 타입(Int)과 같은 기본적인 타입들의 값을 무작위로 만드는 함수들을 제공한다. 예를 들어 1부터 100이내의 정수 값을 무작위로 만들어내는 Haskell 프로그램은 sample (choose (0,100))이다. 이를 실행하면 다음과 같은 결과를 얻는다.

```
GHCi> sample (choose (0, 100))
56
82
86
...
```

GHCi는 Haskell 실행 환경으로 "GHCi>"는 프롬프트 메시지이다. QuickCheck 라이브러리에서 sample과 choose 함수를 제공한다. choose는 어떤 seed를 받아 0에서 100까지의 범위 내에서 임의의 값을 선택하는 함수이고, sample은 초기 seed를 choose함수에 제공해서 무작위 정수 값들을 얻는 함수이다. 위의 예에서 56, 82, 86은 이 과정을 통해 만든 정수 값들이다.

두 번째로 선택적 타입에 대해 랜덤 함수를 정의하는 방법을 설명한다. 불(Bool) 타입과 같이 선택적인 값을 갖는 타입을 Haskell에서 다음과 같이 표현한다.

```
data Bool = True | False
```

타입 선언을 위한 키워드 data를 사용해 Bool 타입을 정의한다. True나 False는 Bool 타입의 참과 거짓을 나타내는 값이다. "|"는 선택 가능한 값들을 나열할 때 사용한다.

불 타입의 값을 무작위로 만드는 Haskell 프로그램은 sample (elements [True, False])이다.

```
GHCi> sample (elements [True, False])
False
True
True
...
```

"["과 "]"는 리스트를 표현하는 것으로 [True, False]는 True와 False 두 가지 값을 원소로 하는 리스트이다.

QuickCheck 라이브러리 함수 elements는 주어진 리스트에서 임의의 원소를 선택하는 함수이다. 위 프로그램을 실행한 결과 처음에는 False를 두 번째와 세 번째는 True를 선택했다.

마지막으로 여러 타입들을 조합해서 정의한 구조화 타입에 대한 값을 무작위로 만드는 함수를 정의하는 방법을 설명한다. 정수와 불을 원소로 갖는 구조화 타입은 다음과 같이 선언한다.

```
data Pair = Pair Int Bool
```

타입 Pair는 데이터 컨스트럭터 (data constructor) Pair로 정수 (Int)와 불 (Bool)을 원소로 하는 구조화 데이터를 값으로 하는 집합이다. 동일한 이름 Pair를 타입과 데이터 컨스트럭터로 혼용해서 사용하는데 이는 문맥에 따라 구분할 수 있기 때문이다. Pair 타입의 값을 임의로 생성하는 함수 genPair는 다음과 같이 정의할 수 있다.

```
genPair = do i <- choose (1,100)
             b <- elements [True, False]
             return (Pair i b)
```

함수 genPair는 먼저 1부터 100까지 정수들 중 하나를 선택해서 변수 i에 놓고, True와 False 중 하나를 선택해서 변수 b에 놓은 후 이 두 값을 가지고 구조화 데이터 "Pair i b"를 만든다. 이 함수를 사용하여 임의의 Pair 데이터를 아래와 같이 만들 수 있다.

```
GHCi> sample genPair
Pair 45 True
Pair 99 False
Pair 33 True
...
```

지금까지 설명한 바와 같이 QuickCheck 라이브러리는 Haskell 타입의 구조에 따라 랜덤 함수를 체계적으로 정의하는 방법을 제공한다. QuickCheck 라이브러리에 대한 상세 설명은 [3]과 [7]을 참고한다. 3GPP 스펙에서 정의하는 SMS PDU 구조를 Haskell 타입으로 표현할 수 있다면 QuickCheck 라이브러리를 활용해서 이 타입의 값을 생성할 수 있을 것이다.

### 3.2 테스트 결과 자동 판단 방법

테스트 결과를 판단하는 문제는 일반적으로 오라클 (Oracle) 문제로 알려져 있다. 예를 들어 테스트 대상 모듈과 동일한 기능을 수행하는 참조 프로그램이 있어서 테스트 결과를 비교할 수 있다면 이 참조 프로그램이 오라클 역할을 한다.

본 논문에서 다루는 SMS PDU 부호화 복호화 모듈에 관한 테스트 문제에서는 부호화 모듈과 복호화 모듈이 서로 역함수인 성질을 이용해 테스트 결과를 판단하는 기준으로 정할 수 있다. 즉 아래의 등식이 성립한다.

```
smspdu = encode (decode smspdu)
```

smspdu는 3GPP 스펙에서 정의한 SMS PDU이다. decode와 encode 함수는 각각 테스트 대상  부호화 모듈과 복호화 모듈이다. decode는 3GPP 스펙에서 정의한 SMS PDU 형식의 바이트들을 받아서 부호화 모듈에서 정의한 형식의 데이터로 변환하는 함수이다. 반대로 encode는 복호화 변환 함수이다.

### 3.3 테스트 범위 (Coverage)

주어진 테스트 방법을 적용했을 때 테스트 대상 모듈을 어느 정도 테스트했는지 확인하는 것은 중요하다.

QuickCheck 라이브러리는 기본적으로 랜덤 테스트 데이터 생성 방법이다. 따라서 테스트 대상 모듈을 모두 테스트함을 항상 보장하지는 않는다. 그럼에도 불구하고 기존의 연구들에서도 테스트 비용 대비 커버리지 비율에 있어서 랜덤 테스트 방법이 매우 효과적이라 주장되어 왔다 [4,5,6].

본 논문에서 제안한 방법은 랜덤 테스트와 블랙 박스 (black box) 테스트이다. 이는 개발 과정 중 주기적으로 수행하는 기본 테스트 (Sanity Test 또는 Smoke Test)로 사용할 수 있다.

## III. SMS 부호화 복호화 모듈 검증 방법

3GPP 스펙에 맞는 SMS PDU를 자동으로 생성해 SMS 부호화 복호화 모듈을 테스트 하는 방법을 제안한다. 리눅스 모바일 플랫폼의 SMS 모듈에 이 방법을 적용하여 적은 비용으로 중요한 오류들을 발견할 수 있었다.

### 1. QuickCheck 라이브러리 기반 SMS PDU 자동 생성 방법

#### 1.1 함수형 언어로 표현한 3GPP SMS PDU 타입

3GPP 스펙에서 정의한 SMS PDU 형식을 함수형 언어 Haskell 타입으로 표현하면 다음과 같다[11].

```
data SMS_PDU =
      SMS_Submit TP_RD TP_VPF TP_RP
          TP_UDHI TP_SRR TP_MR TP_DA TP_PID
          TP_DCS (Maybe TP_VP) TP_UD
    | SMS_Deliver TP_MMS TP_RP TP_UDHI TP_SRI
          TP_OA TP_PID TP_DCS TP_SCTS TP_UD
    | ...
```

타입 SMS_PDU는 3GPP 스펙에서 정의하는 단문 SMS 메시지 포맷을 표현한다. 비록 이 타입은 복잡해 보이지만 <3.1절>에서 설명한 기본 타입, 선택적 타입, 구조화 타입을 조합하여 만든 것이다. SMS_Submit는 송신 SMS 메시지를, SMS_Deliver는 수신 SMS 메시지를 표현한다. 송신 메시

지의 경우 TP_RD, TP_VPF, TP_RP, TP_UDHI, TP_SRR, TP_MR, TP_DA, TP_PID, TP_DCS, (Maybe TP_VP)[2], TP_UD 타입의 값을 포함하고 있다. 각 타입들은 3GPP에서 정의한 송신 SMS 메시지 세부 필드들에 해당한다.

SMS_Submit의 첫 번째 필드 TP_RD 타입은 아래와 같이 선언되어 있다.

```
data TP_RD = TP_RD Bool
```

따라서 이 필드 값은 TP_RD True이거나 TP_RD False이다. 각 필드 타입에 대한 자세한 내용은 [11]을 참고한다.

TP_DCS는 3GPP 스펙에서 정의한 데이터 코딩 방법(Data Coding Scheme)을 표현한 타입이다. TP_DCS의 값에 따라 SMS 메시지를 GSM 7비트, 일반 8비트, 유니 코드 2바이트 (UCS2) 등으로 표현한다.

TP_UD는 사용자 데이터 (User Data)를 표현하는 타입이다. 실제 송수신하는 SMS 메시지 텍스트이다. TP-UD 필드를 함수형 언어 Haskell에서 표현하려면 다음과 같이 선언한다.

```
data TP_UD = TP_UD_NoUDH Int String
           | ...
```

TP_UD_NoUDH는 사용자가 입력한 텍스트 길이와 텍스트로만 구성되어 있고 특별한 헤더를 포함하지 않는 SMS 메시지를 표현한다. 이들을 각각 Int 타입과 String 타입으로 표현한다. 예를 들어 (TP_UD_NoUDH 3 "You")는 SMS 텍스트 메시지 "You"를 표현한 TP-UD 타입의 값이다.

<그림 1>에서 소개한 바이트 포맷 SMS-SUBMIT PDU를 SMS_PDU 타입의 값으로 표현하면 <그림 4>와 같다. 각 심볼의 의미는 [11]에서 정의한 타입 선언을 참고한다.

### 1.2 QuickCheck 라이브러리 기반 SMS PDU 자동 생성 방법

앞 장에서 소개한 SMS_PDU 타입은 3GPP에서 정의한 SMS PDU의 포맷을 표현하도록 정의했다. SMS PDU 타입은 비록 복잡하지만 Int와 같은 기본 타입, Bool과 같은 선택적 타입, Pair와 같은 구조적 타입을 조합해서 선언한 것이다. <II 장>에서 소개한 QuickCheck 라이브러리를 활용하면 기본 타입, 선택적 타입, 구조적 타입을 조합해서 선언한 타입에 대한 값을 무작위로 만들어내는 함수를 정의할 수 있다. 따라서 SMS_PDU 타입에 대한 값을 무작위로 만들어 내는 함수를 QuickCheck 라이브러리를 이용해서 정의할 수 있다. 이렇게 정의한 SMS PDU 자동 생성 프로그램은 [11]과 같다.

SMS PDU 자동 생성 함수를 정의할 때 문제점은 주어진

```
SMS_Submit
 (TP_RD True)
 TP_VP_Field_Present_Absolute
 (TP_RP True)
 (TP_UDHI False)
 TP_SRR_Status_Report_Requested
 (TP_MR 122)
 (TP_DA
  (AddrFields
   (AddrLen 8)
   (TypeOfAddr
    TON_Alphanumeric_gsm7bit
    NPI_Service_center_specific_plan2)
   (AddrValue "a01b*7a2")))
 (TP_PID
  TP_PID_Bits7_6_00
  (TP_PID_Bit5_Bits4_0
  TP_PID_Bit5_Telematic_interworking
  TP_PID_Bits4_0_Specific_to_SC6))
 (TP_DCS_General
  False False Class3 GSM7Bit)
 (Just
  (TP_VP_Absolute 59 11 15 12 12 41 33))
 (TP_UD_NoUDH 3 "You")
```

그림 4. Haskell로 표현한 SMS PDU
Fig. 4. A Haskell representation of SMS PDU

SMS 메시지의 각 필드들을 서로 독립적으로 생성하면 무의미한 조합을 포함하는 SMS PDU를 생성할 경우도 있다. 즉 SMS PDU의 두 필드들이 서로 연관성을 지닌다면 이를 반드시 고려해서 각 필드 값을 만들어야 한다. 예를 들어 TP-DCS 필드에서 GSM 7비트 방식을 사용한다고 지정되었다면 TP-UD 필드는 GSM 7비트 방식을 사용하는 텍스트이어야 한다.

QuickCheck 라이브러리는 "suchThat" 함수를 제공하여 먼저 값을 무작위로 생성한 다음 이 값이 주어진 조건에 맞는지 확인할 수 있도록 한다. 조건에 맞는 경우 최종적으로 그 값을 생성하고 조건에 맞지 않으면 다시 값을 생성한다.

SMS 송신 메시지의 PDU를 임의로 생성하는 Haskell 함수는 <그림 5>와 같이 정의 한다. 전체 코드는 [11]에 있다.

<그림 5>의 함수에서 각 라인은 송신 SMS PDU의 각 필드 값을 차례로 생성한다. 맨 마지막 return문에서 이렇게 만든 모든 필드 값을 구조화 타입 데이터 컨스트럭터 SMS_Submit로 표현한 SMS_PDU 값을 최종적으로 만든다.

* "Maybe TP_VP"는 TP_VP 타입의 값이 옵션임을 뜻한다.

```
genSmsSubmit =
 do tp_rd <- valid :: Gen TP_RD
    tp_vpf <- valid :: Gen TP_VPF
    tp_rp <- valid :: Gen TP_RP
    tp_udhi <- valid :: Gen TP_UDHI
    tp_srr <- valid :: Gen TP_SRR
    tp_mr <- valid :: Gen TP_MR
    tp_da <- valid :: Gen TP_DA
    tp_pid <- valid :: Gen TP_PID
    tp_dcs <- valid :: Gen TP_DCS
    maybe_tp_vp
      <- valid :: Gen (Maybe TP_VP)
            `suchThat` (...)
    tp_ud <- valid :: Gen TP_UD
            `suchThat`(...)
    return (SMS_Submit tp_rd tp_vpf
            tp_rp tp_udhi tp_srr
            tp_mr tp_da tp_pid
            tp_dcs maybe_tp_vp tp_ud)
```

그림 5. 송신 메시지의 SMS PDU를 생성하는 함수
Fig. 5. An SMS-Submit Message Generator

예를 들어 “tp_dcs <- valid :: Gen TP_DCS”는 TP-DCS 필드의 타입 TP_DCS 값을 함수 valid로 무작위로 생성해 tp_dcs에 놓는다.

다음과 같이 위 프로그램을 실행하면 송신 메시지의 SMS PDU를 무작위로 만들 수 있다.

GHCi> sample genSmsSubmit
SMS_Submit (TP_RD True) ...
SMS_Submit (TP_RD False) ...
...

참고로 “tp_ud <- valid :: TP_UD `suchThat` (...)”에서는 TP-UD 필드의 타입 TP_UD 값 tp_ud를 만든 다음 앞서 생성한 tp_dcs의 데이터 코딩 방법을 사용했을 때 정의된 메시지 최대 길이를 넘지 않는지를 검사한다.

## 2. 리눅스 모바일 플랫폼의 SMS PDU 부호화 복호화 모듈 테스트 결과

### 2.1 테스트 대상 SMS 모듈과의 연동

본 논문에서 제안한 테스트 방법을 리눅스 모바일 플랫폼의 메시징 프레임워크 (Messaging Framework)에 포함된 SMS PDU 부호화 복호화 모듈에 적용하였다. 참고로 이 모듈은 C언어로 작성되어 있다.

QuickCheck 라이브러리는 Haskell로 작성되어 있으므로 모바일 디바이스 상에서 직접 테스트하려면 Haskell 라이브러리를 모바일 디바이스에서 실행할 수 있어야 한다. 일반적으로 ARM 프로세서 기반 모바일 디바이스에 Haskell 런타임 시스템을 포팅하는 작업은 상당한 시간을 요구 한다.

따라서 본 논문에서는 간접 테스트 방법을 사용한다. 앞 절에서 제안한 방법으로 생성한 SMS PDU를 C언어로 작성된 테스트 Stub코드의 char 배열로 변환한다. 이 Stub코드를 메시징 프레임워크의 SMS PDU 해석 모듈과 연결해서 모바일 디바이스 상에서 테스트하도록 구성했다.

<그림 4>에서 보여준 함수형 언어 Haskell 타입으로 표현된 SMS PDU를 바이트 형식으로 부호화한 다음 이를 C언어로 표현하면 다음과 같다.

```
int sms_pdu_len = 21;
char sms_pdu[]= {
0xBD, 0x7A, 0x08, 0xD6, 0x0C, 0xD1, 0x7B,
0x2C, 0x3D, 0x03, 0x95, 0x11, 0x51, 0x21, 0x21,
0x14, 0x33, 0x03, 0xD9, 0x77, 0x1D};
```

테스트 Stub 코드는 다음과 같이 테스트 과정을 수행한다. 위의 sms_pdu를 메시징 프레임워크의 SMS 복호화 함수를 통해 이 메시징 프레임워크에서 정의하는 별도의 SMS PDU 자료 구조로 변환한다. 그 다음 이 자료 구조를 메시징 프레임워크의 SMS 부호화 함수를 통해 바이트 단위 SMS PDU로 다시 변환한다. 최종적으로 얻은 SMS PDU와 입력 SMS PDU가 동일하면 해당 SMS PDU에 대해 메시징 프레임워크의 SMS 부호화 복호화 모듈이 잘 동작한다고 판단을 내린다.

### 2.2 테스트 결과

현재 개발 중인 리눅스 모바일 플랫폼의 메시징 프레임워크 SMS 해석 모듈에 100개의 무작위 SMS PDU를 만들어 적용한 결과는 다음과 같다.

표 1. 리눅스 모바일 SMS 모듈에 대한 테스트 결과
Table 1. A Test Result for Linux Mobile SMS Module

| 테스트 결과 | 사례 개수 |
|---|---|
| 성공 | 05개 |
| 버그 | 27개 |
| 미 구현 기능 | 67개 |
| 테스트데이터 오류 | 01개 |

테스트 결과 5가지의 경우 성공했고 27가지의 경우 오류가 발생했다. 자세히 분석해보면 27가지 오류를 실제로 7가지 유형으로 분류할 수 있었다. BCD로 표현된 Time Stamp 값을 잘못 해석하는 경우, 전화번호 내에 포함된 '*'나 '#'과 같은 특수 기호를 처리하지 못하는 경우, 텍스트 길이가 0인 메시지를 처리하지 못한 경우가 있었다. 그 외에 SMS 부호화와 복호화 과정 중 C언어 포인터를 잘못 사용해 발생한 오류들이었다.

특히 서론에서 소개했던 SMS 연도 표시 오류와 유사한 BCD 형식 필드를 잘못 해석한 오류 사례를 본 논문에서 제안한 테스트 방법으로 쉽게 발견할 수 있었다.

미 구현 기능 67가지 사례들은 개발자들이 미처 파악하지 못했던 사항들이었다. 따라서 비록 오류를 발견한 것은 아니지만 개발 진행 중인 메시징 프레임워크의 현재 구현 상황에 대해 파악할 수 있었다. 개발 진행 사항을 점검할 목적으로 주기적인 버전 빌드 후에 제안한 방법을 적용해 볼 수 있을 것이다.

나머지 1가지 오류는 테스트 데이터를 잘못 생성해서 SMS PDU가 3GPP 스펙 정의에 부합되지 않는 경우로 테스트 도구의 오류이다.

### 2.3 테스트 방법의 제약점 및 대응 방안

무작위로 데이터를 만들어 테스트 하는 방법은 대상 범위(coverage)를 모두 빠짐없이 검증함을 보장하지 않는다. QuickCheck 기반 테스트 데이터 자동 생성 방법은 데이터를 무작위로 생성하는 방법이다. 따라서 특정 분포의 테스트 데이터만 생성되는 경우 테스트 대상 모듈의 한정된 범위만 테스트될 것이다.

이 제약점을 보완하기 위해 실행된 소스 범위를 알려주는 툴 (coverage tool) 오픈 소스 gcov를 활용하여 생성한 테스트 데이터로 검증된 소스 범위를 확인해 볼 수 있다. 기준 테스트 범위 (test coverage) 비율을 정해서 이 비율에 이를 때까지 테스트 데이터를 더 많이 만들어 적용함으로써 테스트 범위에 대한 불확실성을 보완할 수 있을 것이다.

QuickCheck 라이브러리 기반 SMS PDU 데이터 생성 프로그램은 두 가지 극단적인 방법으로 데이터를 생성할 수 있다. 랜덤 함수를 이용해서 예상할 수 없는 무작위 값을 생성하거나 특정 유형의 값을 지정해서 그 값만 생성하게 할 수 있다. 두 가지 데이터 생성 유형을 적절히 조절하여 테스트 범위를 조정할 수 있다.

## IV. 결 론

본 논문은 QuickCheck 라이브러리를 활용해 SMS PDU를 자동으로 생성하여 SMS 부호화 복호화 모듈을 테스트하는 방법을 제안했다. 리눅스 모바일 플랫폼의 SMS 해석 모듈에 이 방법을 적용하여 중요한 오류 사례 7가지를 발견하였고 개발자가 쉽게 파악하기 어려운 미 구현 기능들을 찾아냈다. 제안한 방법은 SMS 해석 모듈의 주기적인 Sanity 테스트 방법으로 활용될 수 있을 것이다.

본 논문에서 제안한 SMS PDU 자동 생성 기반 테스트 방법은 3GPP 스펙에서 정의한 PDU 데이터를 생성해서 테스트하므로 이 스펙을 구현하는 모든 SMS 해석 모듈들을 테스트할 때 사용 가능한 장점을 지닌다.

본 논문에서 실증한 바와 같이 스펙이 명확히 정의된 경우에 자동 테스트 데이터를 생성하려면 QuickCheck 라이브러리를 활용하는 방법이 매우 효과적일 것으로 판단된다. SMS 이외의 프로토콜 데이터를 해석하는 모듈을 개발할 때에도 본 논문에서 사용한 방법을 응용해서 해당 프로토콜 데이터 부호화 복호화 모듈을 테스트 가능할 것이다.

향후 연구로 QuickCheck 라이브러리 기반 테스트 자동화 방법을 위해 두 가지를 설명한다. 첫째, 함수형 언어 Haskell로 표현된 SMS PDU를 XML로 변환하는 것이다. C, C++, Java와 같은 주요 프로그래밍 언어들은 XML을 다루는 라이브러리를 모두 제공하므로 테스트 대상 모듈이 이들 언어들로 작성되어 있다면 프로그래밍 언어에 독립적으로 테스트 데이터를 다룰 수 있을 것이다. <그림 4>에서 보인 SMS PDU 예를 보면 알 수 있듯이 Haskell의 SMS_PDU 타입의 값이 XML과 유사한 구조임을 알 수 있다. 참고로 [12]는 XML을 Haskell 타입으로 표현하는 방법을 다룬다.

둘째, 임베디드 소프트웨어 모듈을 Haskell로 작성된 QuickCheck 라이브러리를 연동해 테스트 하는 환경을 개발하는 것이다. 예를 들어 Haskell 실행 환경을 ARM 프로세서 기반 임베디드 디바이스에서도 실행할 수 있도록 경량화 시킬 수도 있다. 혹은 PC상의 Haskell 실행 환경을 USB 등의 인터페이스를 통해 ARM 기반 모바일 디바이스와 연결해 임베디드 소프트웨어를 테스트 할 수 있다.

## 참고문헌


[1] "LG 휴대전화 문자 오류 S/W 업그레이드", 동아일보, *http://news.donga.com/Economy/New/3/01/20100101/25157647/1&top=1*

[2] H.Zhu, P.Hall, and J.May, "Software unit test coverage and adequacy," Computing Surveys, Vol. 29, No. 4, pp366-427, December 1997.

[3] Koen Claessen and John Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," ICFP, pp268-279, ACM SIGPLAN, September 2000.

[4] J.Duran and S.Ntafos, "An evaluation of random testing," Transactions on Software Engineering, Vol. 10, No. 4, pp438-444, July 1984.

[5] D.Hamle, "Random Testing," In J. Marciniak, editor, Encyclopedia of Software Engineering, Wiley, pp970-978, 1994.

[6] R.Hamlet and R.Taylor, "Partition testing does not inspire confidence," Transactions on Software Engineering, Vol. 16, No. 12, pp1402-1411, December 1990.

[7] "QuickCheck:Automatic testing of Haskell programs", *http://www.haskell.org/package/QuickCheck*

[8] The 3rd Generation Partnership Project (3GPP), *http://www.3gpp.org*

[9] 3GPP TS 23.040 V6.7.0, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Technical Realization of the Short Message Service (SMS) (Release 6)", March 2006.

[10] 3GPP TS 23.038 V7.0.0, "3rd Generation partn ership Project; Technical Specification Group Core Network and Terminals; Alphabets and Language-specific Information (Release 7)", March 2006.

[11] *http://hackage.haskell.org/package/GenSmsPdu*

[12] Malcolm Wallace and Colin Runciman, "Haskell and XML: Generic Combinators or Type-Based Translation?," ICFP, pp148-159, March 1999.

[13] 권기창, 백덕화, 권기룡, "역추적 결함 시뮬레이션을 이용한 새로운 테스트 생성 알고리즘," 한국컴퓨터정보학회지, 제 2권, 제 1호, 121-129쪽, 1995년. 6월.

[14] 김경우, "무선인터넷의 현황과 미래 전망에 관한 연구," 한국컴퓨터정보학회지, 제 9권, 제 2호, 19-38쪽, 2002년 6월.


## 저 자 소 개


**최 광 훈**

1996: 한국과학기술원 공학석사.
2003: 한국과학기술원 공학박사.
2006: 토호쿠 대학교 (Tohoku Univ.) 전기통신연구소 (RIEC) 연구원
2006 - 현재: LG전자 책임 연구원
관심분야: 임베디드 시스템, 모바일 시스템, 컴파일러, 프로그래밍 언어, 타입 시스템