사용자 인터페이스를 위한 MVP기반의 XML 언어
(An MVP-based XML Language for User Interfaces)

최 종 명 † 신 경 희 † 유 재 우 ††
(Jong-Myung Choi) (Kyoung Hee Shin) (Chae-Woo Yoo)

요 약 XML을 이용한 사용자 인터페이스 개발은 플랫폼과 프로그래밍 언어에 독립적이며, 사용자가 배우기 쉽고, 사용하기 쉬운 장점을 가지고 있다. 그러나, 현재까지 개발된 XML 기반 사용자 인터페이스 언어들은 정형화된 모델을 사용하지 않고 있으며, 사용자 인터페이스와 내부 로직의 결합 및 이벤트 처리 부분이 미약한 단점을 가지고 있다. 이러한 문제점을 해결하기 위해서 본 논문에서는 MVP (Model-View-Presenter) 모델을 확장한 EMVP (Extended MVP)를 제시하고, EMVP를 기반으로 사용자 인터페이스를 개발할 수 있는 새로운 XML 응용프로그램인 XUIML을 소개한다. XUIML은 EMVP를 기반으로 인터페이스 형태, 이벤트 처리, 데이타 흐름, 인터페이스와 내부 로직의 결합을 기술할 수 있는 방법을 제공한다. XUIML 시스템은 텍스트 편집기와 그래픽 편집기 및 XUIML을 자바와 C# 코드로 변환할 수 있는 변환기를 제공한다. 한편, XUIML의 그래픽 편집기는 직접 조작 방식을 지원하기 때문에 GUI 디자인 도구와 같은 높은 생산성을 얻을 수 있다.
키워드 : XUIML, XML, GUI, 사용자 인터페이스, MVP

Abstract It is advantageous to use XML in developing user interfaces, since XML is independent from platforms and languages and it is easy to learn and use. The existing XML-based languages do not adopt formal model, and they are developed in ad hoc manner. Moreover, they provide limited facilities to handle user events, and combine user interface components with internal logics. In this paper, we introduce an extended MVP (EMVP) model, which is extended from MVP (Model-View-Presenter), and XUIML, which is a new XML application based on the EMVP to support user interface. XUIML provides useful methods to build user interface, including methods to describe actions on user events, data flows between objects, and internal logics. The XUIML system provides two style editors - text editor and graphic editor. It also provides two code generators. One generates Java source, and the other generate C# code from XUIML document. The XUIML graphic editor allows users to manipulate XML elements directly, and we can gain high productivity with the graphic editor.
Key words : XUIML, XML, GUI, user interface, MVP

1. 서 론

컴퓨팅 환경이 발전하면서 GUI(graphic user interface) 사용이 점차 많아지고 있으며, 인터페이스 개발에는 많은 시간과 비용이 소모된다. 응용프로그램에서 인터페이스에 관련된 부분은 전체 코드의 47~60%를 차지하고 있으며[1], 전체 비용의 약 29%를 차지하고 있다[2]. 이렇게 많은 비용과 노력이 소모되기 때문에 인터페이스를 좀더 효과적으로 개발하기 위해서 문법, 태스크 기술 언어[3], 상태도, IRG[2] 등을 이용해서 인터페이스 명세를 기술하기 위한 방법들이 개발되었다. 그러나, 이러한 연구들은 실세계에서 적용하기 어렵고, 효율성이 떨어지기 때문에 널리 사용되지 않는다. 대신에 실세계의 프로그래머들은 쉽게 배우고, 사용할 수 있는 GUI 디자인 도구를 선호한다. 예를 들어, 비주얼 베이직, 델파이 등과 같은 GUI 디자인 도구들이 널리 사용되고 있으며, 이러한 도구를 이용하는 경우에 50~500%의 생산성을 높일 수 있다[4].

그러나, GUI 디자인 도구를 사용하는 경우에 몇 가지

† 학생회원 : 숭실대학교 컴퓨터학과
jmchoi@comp.ssu.ac.kr
khshin@comp.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
cwyoo@comp.ssu.ac.kr
논문접수 : 2002년 1월 7일
심사완료 : 2002년 9월 16일

문제점이 있다. 첫째로, 디자인 도구를 사용하는 경우에 생산성은 높지만, 플랫폼과 특정 언어에 종속적이다. 대부분의 디자인 도구는 특정 플랫폼에서만 작동되고 있으며, 특정 프로그래밍 언어를 기반으로 하고 있다. 예를 들어 비주얼 베이직은 베이직을 기반으로 하고 있으며, 델파이는 파스칼을 기반으로 하고 있다. 따라서 이러한 도구를 사용하는 경우에 주어진 언어 이외에 다른 언어를 사용할 수 없는 문제점이 있다. 둘째로, 디자인 도구를 사용하는 경우에 인터페이스의 레이아웃과 속성은 직접 조작을 통해서 기술하지만, 사용자 이벤트 처리는 프로그래밍 언어를 이용해서 기술하여야 한다. 이러한 경우에 프로그래머가 아닌 일반 사용자는 인터페이스를 작성하기 어렵다. 인터페이스 개발자는 프로그래머뿐만 아니라, 그래픽 전문가, 인간 공학 전문가, 인지 심리학자 등이 될 수 있기 때문에 프로그래머가 아닌 사람도 인터페이스를 개발할 수 있어야 한다. 셋째로, 응용프로그램의 내부 로직과 인터페이스를 결합하기 위해서는 프로그램을 작성하여야 한다. 이것 또한 일반 사용자가 작성하기에는 어려움이 있다.

HTML이 개발된 이후로 웹 기반 응용프로그램들은 HTML을 이용해서 간단하게 사용자 인터페이스를 개발할 수 있게 되었다. HTML로 개발된 인터페이스는 플랫폼 독립적인 장점을 가지고 있다. HTML의 개념을 확장한 XML은 사용자가 원하는 형태의 태그와 속성을 정의할 수 있는 방법을 제공한다. XML[5] 표준이 제정된 이후에 XML을 이용한 데이타 표현[6,7], 프로토콜 개발[7] 및 프로그래밍 개발[8] 등의 연구가 진행됨과 동시에 XML을 이용한 사용자 인터페이스를 개발하려는 연구도 진행되고 있다. XML를 이용해서 인터페이스를 개발하는 가장 중요한 이유는 XML이 플랫폼과 프로그래밍 언어에 무관하고, 배우기 쉽기 때문이다. XUL[9]과 UIML[10,11,12]은 인터페이스 개발을 위한 XML 응용프로그램의 가장 대표적인 예이다.

그러나, 현재까지 개발된 XML 기반 인터페이스 언어는 정형화된 모델을 사용하지 않고, 직관적으로 만들어졌기 때문에 정적인 인터페이스 형태를 구성하는 것은 효과적이지만, 내부 로직과 인터페이스의 결합 및 효과적인 이벤트 처리 방법을 제공하지 못하고 있다. 따라서, XML 기반 언어를 사용하는 경우에도 내부 로직과 결합 및 이벤트 처리를 위한 프로그램을 작성해야 한다. 또한 기존의 XML 기반 인터페이스 언어는 데이타 플로우를 지원하지 못한다. 사용자 이벤트가 발생하는 경우에 인터페이스 컴포넌트간에 혹은 인터페이스 컴포넌트와 내부 로직 사이에 데이타 흐름이 발생한다. 데이타 플로우는 데이타의 이동을 직관적으로 표현하기 때문에 초보자가 프로그램을 쉽게 이해하고, 작성할 수 있도록 도와줄 수 있다.

이러한 문제점들을 해결하기 위해서 본 논문에서는 MVP(Model-View-Presenter) 모델[13]을 확장한 EMVP를 소개하고, EMVP 모델을 기반으로 한 XUIML과 XUIML 구현 시스템을 소개한다. XUIML은 응용프로그램의 사용자 인터페이스 개발을 위한 XML 기반 언어로서 기존의 GUI 디자인 도구나 XML 기반 언어에 비해 몇 가지 장점을 가지고 있다. 첫째로 XUIML은 GUI 디자인 도구들에 비해 플랫폼과 프로그래밍 언어에 독립적이다. XUIML 문서로 기술된 사용자 인터페이스는 코드 생성기를 통해 다른 프로그래밍 언어로 변환될 수 있다. 둘째로 XUIML은 XUL이나 UIML에 비해 간단한 형태로 구성되어 있다. XUIML의 원소와 속성들은 그래픽 인터페이스 컴포넌트의 이름과 속성을 그대로 이용함으로써 사용자가 쉽게 익힐 수 있다. 셋째로 XUIML은 XML만을 이용해서 사용자 인터페이스와 내부 로직의 결합은 물론 이벤트 처리를 기술할 수 있는 방법을 지원한다. XUIML은 특히 이벤트가 발생하는 경우에 객체들간의 데이타 이동을 표현하기 위해 데이타 플로우를 기술할 수 있는 방법을 제공한다. 넷째로 XUIML은 디자인 도구와 같은 높은 생산성을 제공하기 위해서 직접 조작 방식의 그래픽 편집기를 제공한다. 그래픽 편집기는 클릭-포인트 방법을 이용해서 쉽게 사용자 인터페이스를 정의할 수 있는 방법을 제공한다.

논문에서 구현한 XUIML 시스템은 텍스트 편집기와 그래픽 편집기를 제공한다. 텍스트 기반 편집기는 텍스트 형태로 XUIML을 기술하고, 원하는 프로그래밍 언어로 변환해서 인터페이스를 생성할 수 있는 기능을 제공한다. 그래픽 기반 편집기는 인터페이스 디자인 도구와 같이 직접 조작을 통해서 XUIML 문서를 생성할 수 있는 방법을 제공한다. 그래픽 편집기는 디자인 도구와 같이 높은 생산성을 제공할 수 있는 장점이 있다.

본 논문은 2장에서 관련 연구들을 소개하고, 3장에서 MVP와 XUIML에 관한 사항들을 소개한다. 4장에서는 XUIML이 사용되는 형태를 소개한다. 5장에서는 시스템 구성과 XUIML 편집기를 소개하고, 6장에서 결론을 맺는다.

2. 관련 연구

2.1 XUL

XUL은 Mozilla 웹 브라우저[14]를 보다 빠르고, 쉽게 개발하기 위해 설계되었고, 정적인 GUI를 기술하기

위한 XML 기반 언어이다. XUL은 HTML과 다른 XML(예: MathML, SVG)을 포함할 수 있으며, 사용자 인터페이스의 모양(예: 스킨, 윈도우 매니저 등)을 변경하기 위해 스타일 시트를 사용한다. 사용자 인터페이스는 content, skin, locale의 세 부분으로 구성되어 있다. content는 사용자 인터페이스 원소들과 스크립트를 포함하고, skin은 스타일 시트와 이미지들을 포함한다. 스타일 시트는 CSS를 이용하며, 윈도우의 모양을 보다 세밀하게 지정하기 위해서 사용된다. XUL은 이벤트를 처리하기 위해 HTML에서 사용하던 방법과 유사하게 자바 스크립트를 이용한 이벤트 핸들러를 사용한다. XUL은 웹 브라우저를 위해서 개발된 언어이기 때문에 일반 응용프로그램의 인터페이스 개발에는 적합하지 않다.

2.2 UIML

UIML은 주변 장치에 무관하게 사용자 인터페이스를 기술하기 위해 개발되었다. UIML은 타겟 디바이스에서 사용되는 언어(예: HTML, WML, VoiceML, C++, Java 등)에 자동적으로 매핑된다. UIML은 head, interface, peers, template이라는 4개의 원소로 구성되어 있다. head 원소는 문서에 대한 메타 데이타(예: 저자, 버전 등)를 제공한다. interface 원소는 사용자 인터페이스를 구성하는 인터페이스의 구조(structure), 내용(content), 스타일(style), 행동(behavior)을 기술한다. structure 원소는 다양한 플랫폼을 위한 인터페이스와 인터페이스들 간의 구성을 기술한다. style 원소는 인터페이스와 관련된 다양한 속성 값들을 기술한다. content 원소는 사용자에게 전달할 정보(예: 텍스트, 음성, 이미지)를 가지고 있다. behavior 원소는 사용자가 인터페이스 위짓(widget)과 상호 작용할 때 어떤 액션이 수행되어야할지를 기술한다. peer 원소는 UIML의 각 속성과 이벤트 이름을 UI 툴킷과 내부 로직에 매핑시키는 역할을 한다. template 원소는 UIML 문서의 일부분을 재활용하기 위해서 사용된다. 그러나 UIML은 내부 로직과 사용자 인터페이스를 연결하는 부분이 부족하고, 데이타 플로우 기능 및 UIML을 편리하게 개발할 수 있는 그래픽 편집기를 지원하지 않는다.

3. MVP와 XUIML

3.1 사용자 인터페이스와 MVP(Model-View- Presenter)

응용프로그램은 사용자 인터페이스와 프로그램의 내부 로직으로 구성되어 있다. 사용자 인터페이스는 내부 로직의 데이타와 상태를 외부로 표현하고, 사용자와 상호 작용한다. 내부 로직은 응용프로그램의 내부적인 논리 부분을 구현하고, 외부 엔티티(예: 데이타베이스)와 상호 작용한다. 사용자 인터페이스와 내부 로직은 서로 연결되어 있다. 사용자 인터페이스와 내부 로직이 강하게 결합된 경우에는 그림 1과 같이 사용자 인터페이스가 내부 로직을 생성 및 사용하고, 역으로 내부 로직도 사용자 인터페이스를 사용한다. 이러한 경우에는 사용자 인터페이스나 로직의 일부분을 수정해야 하는 경우에 프로그램 전체를 고쳐야 하는 문제가 발생한다.

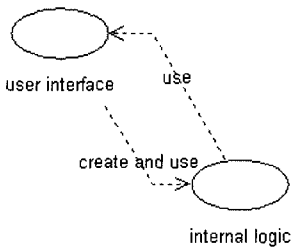

그림 1 강하게 결합된 사용자 인터페이스와 내부 로직

사용자 인터페이스와 내부 로직이 잘 분리되어 있는 경우에는 그림 2와 같은 형태로 구성된다. 사용자 인터페이스는 내부 로직의 상태를 표현하기 위해서 내부 로직을 생성하고, 사용한다. 그러나, 내부 로직은 사용자 인터페이스의 코드를 사용하지 않는다. 이러한 구조에서는 프로그램의 일부를 변경해야 하는 경우에 변경되는 부분을 최소화할 수 있다.

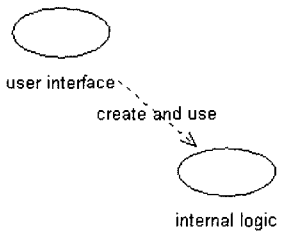

그림 2 느슨하게 연결된 사용자 인터페이스와 내부 로직

느슨하게 연결된 사용자 인터페이스와 내부 로직은 MVP (Model-View-Presenter) 구조[13]로 표현될 수 있다. MVP는 MVC (Model-View-Controller) 구조[15]의 변형된 형태이다. 기본적으로 사용자 인터페이스와 이벤트 처리를 제공하는 현재의 운영체제에서는 MVC 구조 보다는 MVP 구조가 좀더 적합하다. 느슨하게 결합된 인터페이스와 내부 로직을 MVP 형태로 표현하면 그림 3과 같은 형태로 표현할 수 있다. MVP에

서 View와 Presenter는 사용자 인터페이스에 해당되고, Model은 내부 로직에 해당된다. 그림 3에서 Presenter와 View는 Model을 직접 접근할 수 있지만, Model은 Presenter와 View를 직접 접근할 수 없다.

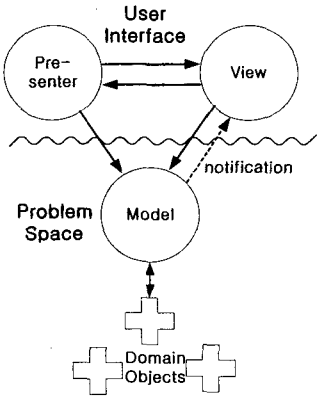

그림 3 MVP 모델

Model은 응용프로그램의 내부 로직과 데이타를 가지는 부분이다. 내부 로직과 데이타는 도메인 객체에 해당된다. Model은 View에 대한 정보를 전혀 가지고 있지 않다. MVP의 Model이 MVC와 다른 점은 MVC 구조에서 Model은 View의 이벤트를 처리하기 위한 함수들을 포함하고 있는데 비해서, MVP의 Model은 View의 이벤트 처리를 위한 메소드를 전혀 포함하고 있지 않다는 것이다[13]. Model에서 값의 변화는 notification에 의해서 간접적으로 전달된다. View는 Model의 데이타와 상태를 화면에 표현하는 역할을 한다. MVC에서 Controller 부분은 MVP에서는 제거되었다. 대신에 View의 인터페이스 위젯(widget)들이 운영체제에서 생산하는 저수준의 이벤트들을 처리한다[13]. Presenter는 Model의 데이타를 관리하고, 사용자 입력에 대해서 어떻게 처리할 것인지를 결정하는 이벤트 처리 함수들로 구성된다.

3.2 EMVP(Extended MVP)

MVP 모델은 현재 GUI 환경에 적합한 인터페이스 모델이지만, XUIML과 같이 XML을 이용해서 사용자 인터페이스의 형태, 이벤트 처리, 인터페이스와 내부 로직의 결합 등을 선언적으로 기술하기 위해서는 MVP를 확장할 필요가 있다. 확장된 MVP를 EMVP (Extended MVP)라고 부르기로 한다. EMVP에서 GUI 응용프로그램은 MVP에 EventListener, ModelListener, Model ViewMap을 추가한 6개의 튜플로 표현할 수 있다.

정의: GUI 응용프로그램

GUI 응용프로그램 = <Model, View, Presenter, EventListener, ModelListener, ModelViewMap>

GUI 응용프로그램은 내부 로직을 위한 Model, 사용자 인터페이스 위젯을 위한 View, 이벤트 처리를 위한 Presenter와 MVP 요소들을 연결하기 위한 Event Listener, ModelListener, ModelViewMap를 포함한다.

정의: Model

Model은 도메인 객체들의 집합이다. 도메인 객체는 속성(데이타)과 내부 로직 함수들을 가지고 있다. m_i(∈ Model)의 속성 a는 m_i.a로 표현하고, 함수 b()는 m_i.b()로 표현한다.

정의: View

View는 GUI 응용프로그램의 인터페이스 위젯들의 집합이다.

사용자 인터페이스는 가장 상위의 윈도우를 중심으로 포함 관계에 의해서 트리 형식으로 구성된다. 사용자 인터페이스는 다른 인터페이스 위젯을 포함할 수 있는 컨테이너(container) 타입과 그렇지 않은 프리미티브(primitive) 타입으로 분류할 수 있다. 프리미티브 타입은 인터페이스 트리의 단말 노드를 구성하고, 컨테이너 타입의 위젯들은 트리의 중간 노드를 형성한다. 컨테이너 위젯들은 자신이 포함하고 있는 서브 인터페이스 위젯의 레이아웃을 관리하기 위한 속성을 가지고 있고, 서브 인터페이스 위젯은 자신의 위치를 지정하기 위한 속성을 가지고 있다. 또한 모든 인터페이스 위젯 u_i(∈ View)는 자신을 식별할 수 있는 id와 자신을 포함하는 컨테이너에 대한 레퍼런스 속성(parent) 및 기타 인터페이스 관련 속성들을 가지고 있다.

프리미티브 타입의 위젯들은 가지고 있는 값에 따라 세 부류로 구분할 수 있다. 첫째는 값을 갖지 않는 위젯이다. 예로 라벨, 버튼은 값을 갖지 않는 위젯들이다. 둘째는 하나의 값을 갖는 위젯들이다. 예로는 텍스트 필드, 슬라이더, 스피너(spinner), 체크박스 등이 있다. 값이 하나인 위젯들은 value 속성을 가지고 있다. 셋째로는 여러 개의 값을 가질 수 있는 위젯들이다. 예로는 리스트, 콤보박스, 테이블, 트리 등을 들 수 있다. 값이 여러 개인 위젯들은 값을 추가하기 위한 add(), 삭제하기 위한 remove(), 값을 선택하기 위한 select() 함수들을 갖는다. 또한 선택된 값을 위한 select 속성을 가지고 있다.

Presenter는 사용자 혹은 시스템이 발생시킨 이벤트 혹은 Model의 상태 변화를 처리하는 함수들의 집합이다.

정의: Presenter

Presenter = { p_i | p_i : (s_i, e_i) ↦ (v_i, m_i),
$s_i \in$ (Model ∪ Primitive), $e_i \in$ Event,
$v_i \in$ View, $m_i \in$ Model }

이벤트 처리 함수 p_i는 사용자 이벤트 혹은 내부 로직의 상태 변화에 의해서 호출될 수 있다. 이벤트 처리 함수는 인터페이스 위짓과 내부 로직의 상태를 변화시킨다. 때로는 이벤트 처리 함수들은 사용자 인터페이스와 내부 로직 사이의 번역기(translator) 역할을 한다. 예를 들어 사용자 인터페이스에서 입력된 값들은 내부 로직에서 사용되는 값의 형태로 번역되어서 내부 로직에 전달된다. 역으로 내부 로직의 값들은 사용자 인터페이스에 맞게 번역되어야 한다. 이벤트 처리 함수는 또한 데이타 흐름을 유도한다. 데이타 흐름은 사용자 인터페이스 간에 혹은 모델과 사용자 인터페이스 간에 발생할 수 있다.

이벤트 드리븐 프로그래밍에서 이벤트를 처리하기 위해서는 사용자 위짓에서 관심있는 이벤트를 등록하여야 한다. 이벤트를 등록하면, 사용자 위짓에서 관심있는 이벤트가 발생하는 경우에 Presenter의 함수가 호출된다. 이렇게 사용자 위짓, 이벤트, 이벤트 처리 함수를 연결하기 위해서 사용되는 것이 EventListener이다. EventListener는 인터페이스 위짓에 이벤트가 발생하는 경우에 처리할 함수의 매핑들 집합이다.

정의: EventListener

EventListener = { <u_i, e_j, p_k> | $u_i \in$ Primitive,
$e_j \in$ Event, $p_k \in$ Presenter }

Model의 상태 변화는 때로는 View를 통해서 외부에 표현하여야 할 때가 있다. 예를 들어, 통신 프로그램에서 갑작스럽게 네트워크 연결이 끊어지는 경우에 에러 상태를 사용자에게 알려주어야 할 필요가 있다. 이러한 Model의 상태 변화는 사용자의 상호 작용에 무관하게 발생한다. ModelListener는 모델 객체의 상태 변화에 따라 호출되는 Presenter의 함수를 기술하는 매핑이다.

정의: ModelListener

ModelListener = {<p_h, h_i, $m_j.a_k$> |
$p_h \in$ Presenter, $h_i \in$ AE2,
$m_i \in$ Model }

AE2 = Adapter ∪ Event ∪ Exception

Model이 Presenter의 메소드를 직접 호출하는 경우에 인터페이스와 내부 로직의 분리가 이루어지지 않기 때문에 Model은 Presenter의 함수를 직접 호출할 수 없다. 이러한 경우에 Model이 Presenter의 함수를 간접적으로 호출할 수 있는 다음과 같은 세 가지 방법을 찾을 수 있다. 첫 번째 방법은 Model에 어댑터 클래스를 이용하는 것이다. 즉, Model의 상태가 변경되는 경우에 어댑터를 이용해서 View의 상태를 변경하는 방법이다. 두 번째 방법은 사용자 이벤트를 사용하는 것이다. Model의 상태가 변경되는 경우에 사용자 이벤트가 발생해서, Presenter의 함수를 호출하고, Presenter의 함수는 Model의 상태를 읽어서 View에 전달하게 된다. 세 번째 방법은 예외(exception)를 이용하는 것이다. 이것은 Model에서 상태가 변경되는 경우에 사용자가 정의한 예외가 발생함으로써 View의 상태를 변경하는 방법이다. 세 방법 모두 모델의 상태 변화는 Presenter의 함수를 통해서 View의 형태를 변화시킨다.

사용자 인터페이스와 내부 로직은 느슨하게 연결되어 있으면서도, 인터페이스와 내부 로직의 상태는 밀접하게 연결되어 있어야 한다. 즉, Model m_j의 속성 a_k는 u_i에 반영된다. 이때 a_k의 상태가 변경되는 경우에 u_i도 변경되고, u_i가 변경되는 경우에는 m_j의 a_k 상태도 변경된다. 이러한 관계는 ModelViewMap을 통해서 기술할 수 있다. ModelViewMap은 모델의 상태와 사용자 인터페이스의 상태의 매핑이다.

정의: ModelViewMap

ModelViewMap = { <u_i, $m_j.a_k$> | $u_i \in$ View,
$m_j \in$ Model }

MVP에서 인터페이스 위짓은 Model의 외부 함수를 직접 접근할 수 있기 때문에 View의 상태 변경은 직접 Model에 전달될 수 있다. 그러나, Model의 상태 변화는 직접 View에 전달될 수 없다. 때문에 ModelListener에 등록된 Presenter의 함수를 통해서 Model의 상태 변화가 View에 전달된다.

EMVP 모델은 그림 4와 같이 표현될 수 있다. 그림 4에서 실선 화살표는 직접 접근 가능함을 의미하고, 점선 화살표는 간접적인 접근을 의미한다. View의 인터페이스 위짓들은 포함 관계에 의해서 인터페이스 트리를 형성하게 된다. 사용자 이벤트가 발생하는 부분은 트리의 단말 노드에 해당되는 프리미티브 타입이다. 프리미티브 타입에서 사용자 이벤트가 발생하는 경우에 관심있는 이벤트들은 EventListener에 의해서 Presenter의 함수를 호출하게 된다. Presenter의 함수들은 Model의 데이타를 변경하거나, Model의 함수를 호출할 수 있다. 또한 View의 프리미티브 위짓의 상태를 변경하거나 함수를 호출할 수 있다. View는 필요한 경우에 Model의 데이타에 접근하거나, 메소드를 호출할 수 있다. Model은 ModelListener를 통해서 Presenter의 함수를 호출할 수 있다. 또한 ModelViewMap 통해서 View의 프리미티브 위짓의 상태를 변경할 수 있다.

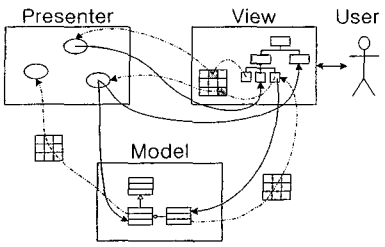

그림 4 EMVP 상세 구조

3.3 EMVP와 XUIML 매핑

EMVP는 사용자 인터페이스와 내부 로직을 분리하면서, 내부 로직과 인터페이스를 느슨하게 연결할 수 있는 방법을 제공한다. XUIML은 EMVP를 지원할 수 있도록 하기 위해서 EMVP를 XML 문서로 매핑시킨다. EMVP의 GUI 응용프로그램을 XML로 표현하기 위해서 그림 5와 같은 DTD 구조로 표현할 수 있다. 그림 5에서 사각형은 원소를 의미하고, 회색으로 채워진 사각형은 엔티티 참조, 둥근 사각형은 속성을 의미한다. XUIML의 가장 상위 원소는 xuiml이다. xuiml 원소는 응용프로그램 이름을 속성으로 가지고 있다. xuiml 원소의 자식 원소는 MVP 구조에 따라 model, view, presenter라는 세 부분으로 구성되어 있다.

응용프로그램의 내부 로직은 특정 프로그래밍 언어로 작성된다. 작성된 내부 로직은 GUI 인터페이스와 연결되어야 하는데 이때 사용되는 것이 model 원소이다. model 부분은 특정 프로그래밍 언어로 구현된 내부 로직과 데이타를 사용자 인터페이스와 연결시키기 위해서 사용된다. model 원소는 인터페이스 프로그램에서 사용될 로직 객체를 변수로 표현하기 위해서 var 자식 원소를 갖는다. 또한 응용프로그램을 초기화하기 위한 init 원소와 응용프로그램 종료시 클린업을 위한 destroy 원소를 가지고 있다.

view 원소는 사용자 인터페이스를 표현하기 위해서 사용된다. 사용자 인터페이스는 가장 상위 윈도우를 표현하기 위해서 frame 원소를 사용한다. frame 원소는 내부에 다른 사용자 위젯들을 포함할 수 있다. 컨테이너 위젯은 원소의 내용으로 다른 위젯들을 포함한다. 이러한 포함 관계는 뷰 트리를 구성하게 된다. 프리미티브 원소는 다른 인터페이스 위젯을 포함할 수 없다.

presenter 원소는 사용자 인터페이스에서 발생하는 이벤트를 처리하기 위해서 사용되는 부분이다. presenter는 이벤트를 처리하는 handler 원소를 가지고 있다. 이벤트를 처리하는 함수를 표현하는 handler 원소는 인터페이스 위젯과 모델의 상태를 알아보고, 변경하기 위한 get과 set 원소를 가지고 있다. 또한 메소드를 호출하기 위한 call 원소와 데이타 흐름을 표현하기 위한 dataflow 원소를 가지고 있다.

EventListener를 기술하기 위해서 XUIML에서는 primitive 타입의 서브 원소로 event 원소를 사용한다. event 원소는 이벤트 타입(type), 이벤트 처리 함수(handler)의 레퍼런스를 지칭하는 속성을 가지고 있다. 이벤트가 발생한 위젯은 event 원소를 포함하는 primitive 원소에 해당되는 인터페이스 위젯이다.

ModelViewMap을 기술하기 위해서 primitive는 model 속성을 가지고 있다. 이것은 내부 로직의 값이 인터페이스를 통해서 외부에 표현되어야 할 때 사용될 수 있다. 인터페이스에서 변경된 값은 내부 로직에 반영되고, 내부 로직에서 변경된 값은 외부 인터페이스에 반영된다. 내부 로직에서 사용되는 값과 인터페이스 위젯에서 사용되는 값이 데이타 타입이 다르거나 포맷이 다른 경우에는 filter 속성을 통해서 변환될 수 있다.

ModelListener를 지원하기 위해서 XUIML에서는 이벤트와 어댑터 클래스를 이용하는 방법을 지원한다. model의 var 원소는 서브 원소로 event를 가지고 있다.

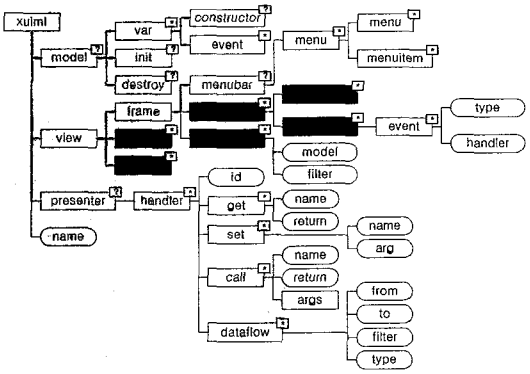

그림 5. XUIML의 DTD 구조

4. XUIML 구성

4.1 사용자 인터페이스 형태

사용자 인터페이스의 형태는 view 원소를 통해서 결정된다. view 원소는 사용자 인터페이스 위젯에 따라 XML 원소들을 포함한다. 인터페이스에서 컨테이너는 다른 위젯 원소를 서브 원소로 포함할 수 있다. 각 인터페이스 위젯의 특성은 XUIML 원소의 속성으로 표현된다. 위젯을 위한 모든 XUIML 원소들은 유일하게 식별

할 수 있는 id 속성을 가지고 있다. 인터페이스 위젯에 해당되는 각 XUIML 원소들을 각 위젯에 해당되는 특성을 표현하기 위한 속성들을 가지고 있다. 그러나, 프로그래밍 언어와 플랫폼에 독립적이어야 하기 때문에 사용할 수 있는 속성들이 제한적이다. 표 1은 view 원소를 사용하는 예를 보여준다. view 원소 내부에는 인터페이스 위젯을 위한 원소들이 올 수 있다. 표 1에서 textarea는 텍스트 영역의 위치를 기술하기 위해서 position 속성을 포함하고 있다. 또한 이벤트를 처리하기 위해서 fileQuit 메뉴 아이템은 event 서브 원소를 포함하고 있으며, presenter의 handler와 연결되어 있다. 따라서, fileQuit 메뉴 아이템을 클릭하는 순간에 프로그램을 종료하게 된다.

표 1 view 원소 예

```
<view>
   <frame title="XUIML" size="300,200" layout="position">
      <menubar>
         <menu text="File">
            <menuitem id="fileNew" text="New"/>
            <menuitem id="fileClose" text="Close"/>
            <menuitem id="fileQuit" text="Quit">
               <event type="action" handler="exit"/>
            </menuitem>
         </menu>
      </menubar>
      <textarea id="area" position="5, 5, -5, -5"/>
   </frame>
</view>
<presenter>
   <handler id="exit">
      <call name="xuiml:exit"/>
   </handler>
</presenter>
```

표 1의 XUIML 문서는 XUIML2SWING 코드 생성기를 통해 그림 6과 같은 사용자 인터페이스를 생성한다.

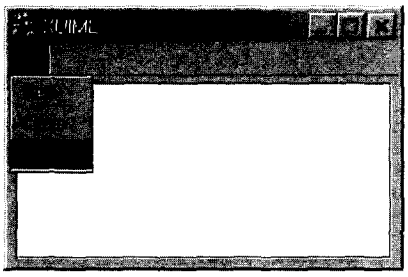

그림 6 생성된 인터페이스 예

4.2 내부 로직과 연결

내부 로직의 클래스와 데이타들은 사용자 인터페이스에서 생성되고 사용된다. model은 var, init, destroy 서브 원소를 가지고 있다. init과 destroy는 응용프로그램의 초기화 작업과 종료시 필요한 작업들을 기술할 수 있는 부분들이다. model은 내부에 var 원소를 가지고 있으며, 속성으로 id와 type을 가지고 있다. id는 유일하게 식별하기 위한 속성이고, type은 변수의 데이타 타입을 기술한다. 데이타 타입이 기본형이 아닌 경우에는 constructor 원소를 이용해서 생성자를 호출하기 위해 필요한 매개 변수들을 기술한다.

표 2는 model 원소의 사용 예를 보여준다. model의 var 원소는 id 속성의 값이 "hong"으로 지정되어 있고, type 속성은 "Employee"이다. Employee는 응용프로그램의 내부 데이타를 표현하기 위해서 사용되는 것으로, 내부 로직 개발자에 의해서 프로그래밍 언어를 이용해서 개발된다. 인터페이스 개발자는 내부 로직 개발자가 구축한 Employee 데이타를 사용하기 위해서 model 원소를 이용한다.

표 2 model 원소 예

```
<model> .....
   <var id="hong" type="Employee">
      <constructor>
         "Gil D. Hong", 29, "male"
      </constructor>
   </var> .....
</model>
```

XUIML로 표현된 표 2의 내용은 자바 언어로 표현하는 경우에 다음과 같이 변경될 수 있다.

```
Employee  hong = new
Employee("Gil D. Hong", 29, "male")
```

model 원소에서 생성된 데이타는 view와 presenter에서 사용될 수 있다. EMVP에서 MVMap은 프리미티브 인터페이스 위젯의 model 속성을 통해서 구현된다. 표 3은 model 원소를 통해서 생성된 내부 로직 데이타가 인터페이스의 위젯의 model 속성을 통해서 매핑되는 것을 보여준다. id 속성이 "hong"인 Employee 타입의 데이타는 트리를 나타내는 view 부분에서 트리 노드의 데이타를 표현하기 위해서 사용된다.

표 3 model 데이타의 사용

```
<view> ...
    <tree> ...
      <node model="dept">
        <node id="nodeHong" model="hong"/>
        <node id="nodeKim"  model="kim"/>
      </node>
    </tree> ...
</view>
```

Model은 여러 개의 View에 매핑될 수 있다. 그림 7은 하나의 Model 데이타가 여러 개의 사용자 위젯으로 표현되는 것을 보여준다. Model의 Media.volume 속성은 View의 슬라이더와 스핀너로 매핑된다. 슬라이더에서 변경된 값은 Media.volume 속성 값을 변경시키고, 이것은 다시 스핀너에 표현되는 값을 변경하게 된다.

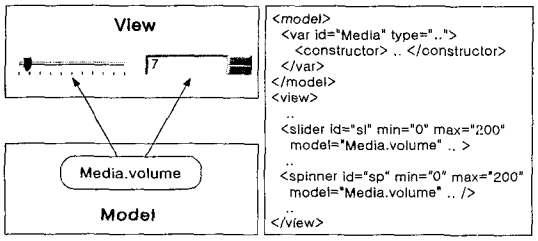

그림 7 Model-View 매핑

4.3 사용자 이벤트 처리

presenter 원소는 사용자 이벤트를 처리하기 위한 부분이다. 사용자 이벤트에 의한 액션은 크게 네 가지 형태로 분류할 수 있다.

- 상태 파악
- 상태 변경
- 메소드 호출
- 데이타 이동

상태 파악과 상태 변경은 인터페이스 위젯 혹은 model 데이타의 상태를 알아보는 getXXX() 메소드와 상태를 변경시키는 setXXX() 형태의 간단한 메소드 호출이다. 예를 들어 버튼의 활성화 상태를 변경하는 것은 대표적인 상태 변경 액션이다. 메소드 호출은 model의 함수와 위젯의 함수를 호출하는 것이다. 이것은 상태를 변경하는 것이 아니라, 어떤 작업을 수행하도록 지정하는 것이다. 데이타 이동은 위젯-위젯, model-model, model-위젯 간에 데이타를 주고받는 것이다.

그림 8은 XUIML에서 사용자 이벤트를 처리하는 방법을 보여준다. "->" 라벨이 있는 버튼에서 action 이벤트가 발생하는 경우에 addItem 이벤트 핸들러 함수가 호출된다. addItem 이벤트 핸들러는 우측의 리스트의 add 메소드(right.add)를 호출하고, 매개 변수로 좌측의 선택된 아이템(left.select)을 전달한다. 마지막으로 좌측 리스트의 remove 메소드(left.remove)를 호출해서 좌측 리스트에서 선택된 아이템을 삭제한다.

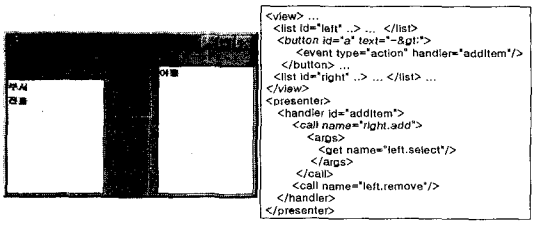

그림 8 XUIML에서 이벤트 처리

4.4 데이타 흐름

presenter 원소 내부에는 데이타의 흐름을 기술하기 위한 dataflow 원소를 기술할 수 있다. dataflow는 데이타 흐름을 유일하게 식별할 수 있는 id와 데이타의 타입을 기술할 수 있는 type 속성을 가진다. dataflow는 또한 데이타의 원천과 목적지를 기술하기 위해서 from과 to 속성을 가진다. 그림 8에서 좌측 리스트에서 우측 리스트로 데이타가 이동되는 것은 표5와 같이 변경해서 표현할 수 있다.

표 5 dataflow 원소 예

```
<dataflow id="toLeft" type="char" from="left" to="right">
    <value name="left"/>
</dataflow>
```

데이타 흐름은 위젯-위젯, model-model, model-위젯 간에 발생할 수 있다. 이러한 데이타흐름의 지원은 프로그램을 보다 쉽고, 직관적으로 작성할 수 있도록 한다.

5. 시스템 구성과 XUIML 편집기

XUIML를 구현한 시스템은 그림 9와 같은 구조로 구성되어 있다. 사용자는 텍스트 편집기나 XUIML 전용 그래픽 편집기를 이용해서 XUIML을 작성한다. 작성된 XUIML 문서는 XUIML 파서에 의해 문법적인 오류를 체크한다. 문법적인 오류가 없는 XUIML 문서는 자바 혹은 C# 언어 코드 생성기를 통해서 해당 언어 코드를 생성한다. 코드 생성기는 내부적으로 코드 변환 정보를 위한 규칙 시스템을 사용한다. 규칙 시스템은 XUIML

원소와 속성을 해당 언어로 매핑해주기 위한 규칙들이 포함되어 있다.

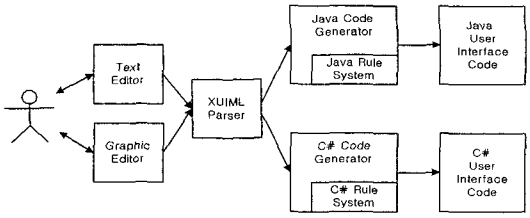

그림 9 시스템 구조

그림 10은 XUIML 텍스트 편집기이다. XUIML 텍스트 편집기는 XUIML 문서를 작성해서 프로그래밍 언어로 컴파일하고, 실행시킬 수 있는 기능을 제공한다.

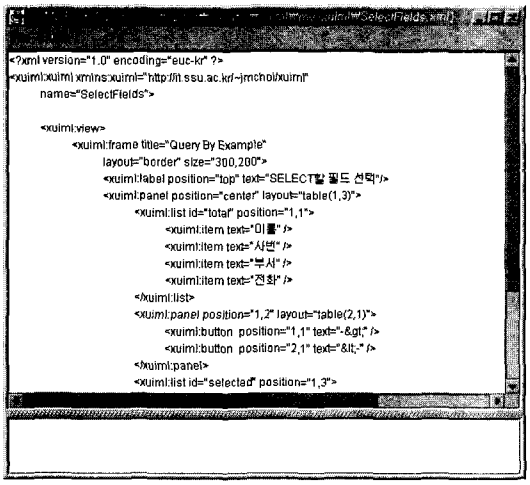

그림 10 XUIML 텍스트 편집기

그림 11은 XUIML을 위한 그래픽 편집기이다. 그래픽 편집기는 위짓 컴포넌트를 직접조작 방식으로 인터페이스를 작성할 수 있고, 속성을 보다 쉽게 변경할 수 있는 기능을 제공한다. XUIML 그래픽 편집기를 사용하는 경우에 GUI 디자인 도구를 사용하는 것과 같은 높은 생산성을 기대할 수 있다.

XUIML 그래픽 편집기는 정적인 사용자 인터페이스 위짓들을 화면에 표현할 뿐만 아니라, 인터페이스의 동적인 내용도 화면에 모두 표현한다. 예를 들어 그림 11에서 사용자가 File 메뉴를 클릭하면, New, Open 등의 메뉴 아이템들이 나타나고, 사용자가 다시 Open 메뉴 아이템을 선택하면, 파일 다이얼로그가 나타난다. 파일 다이얼로그에서 사용자가 파일을 선택하는 경우에 데이타 흐름을 통해서 파일 내용이 텍스트 영역에 나타난다.

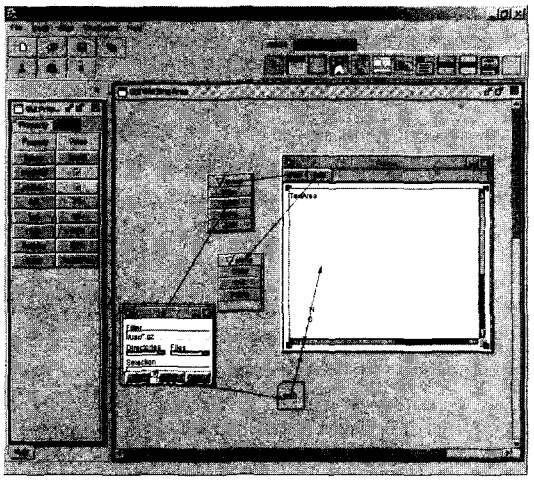

그림 11 XUIML 그래픽 편집기

6. 결 론

현재 XUL과 UIML과 같은 XML 기반의 인터페이스 언어는 주로 웹 도메인에 맞추어져 있다. 그러나, XML 기반의 인터페이스 언어는 플랫폼과 프로그래밍 언어에 독립적이기 때문에 다른 분야에서도 많이 사용될 것이다. 특히 XML은 프로그래밍 언어에 비해 배우기 쉽고, 사용하기 쉽기 때문에 프로그래머가 아닌 일반 사용자들도 쉽게 사용자 인터페이스를 작성할 수 있다.

본 논문에서는 XUIML이라는 EMVP 모델을 기반으로 설계된 사용자 인터페이스용 XML 언어를 소개하였다. XUIML은 EMVP 구조에 따라 model, view, presenter 부분으로 구성되어 있다. model 부분은 사용자 인터페이스에서 내부 로직을 접근하기 위한 변수들을 선언하고, view는 사용자 인터페이스의 형태와 구조를 결정한다. presenter는 사용자 이벤트에 따른 액션을 기술한다. XUIML은 다른 XML 기반 인터페이스 언어와는 달리 정형화된 모델을 기반으로 작성되었기 때문에 내부 로직과 인터페이스 결합, 효과적인 이벤트 처리, 데이타 흐름을 지원하는 장점을 가지고 있다.

XUIML은 프리젠테이션과 내부 로직을 분리하도록 함으로써, 개발자와 디자이너의 역할을 명확히 구분하도록 도와준다. 따라서, XUIML을 이용하는 경우에 인터페이스 디자이너는 인터페이스 디자인에만 정신을 집중할 수 있고, 프로그래머는 내부 로직만 작성할 수 있다.

본 논문에서 구현한 XUIML 시스템은 텍스트 기반 편집기와 그래픽 기반 편집기를 제공한다. 또한 XUIML을 자바와 C# 언어로 변환할 수 있는 코드 생성기를 제공한다. XUIML을 이용한 인터페이스 개발은 개발 기간과 비용을 감소시켜줄 것이다.

참 고 문 헌

[1] MacIntyre F., Estep K.W., Sieburth J. M., "Cost of user-friendly programming," in *Journal of Forth Application and Research*, 6(2), pp.103-115, 1990.

[2] Rosenberg, D., "Cost-benefit analysis for corporate user interfaces standards: What price to pay for a consistent look and feel," in *Coordinating user interfaces for consistency*, New York Academic Press, pp.21-34, 1989.

[3] Payne S.J., Green T.R.G., "Task-action Grammars: A model of the mental representation of task languages," in *Human Computer Interaction*, Vol. 2, pp.93-133, 1986.

[4] Ben Shneiderman, *Designing the User Interface Strategies for Effective Human-Computer Interaction*, Addison Wesley, 1998.

[5] XML, http://www.w3.org/xml/

[6] XML Schema, http://www.w3.org/XML/Schema/

[7] Simple Object Access Protocol, http://www.w3.org/TR/SOAP/

[8] G. Badros, "JavaML: A markup language for Java source code," in *Computer Networks*, Vol. 33, pp. 159-177, June, 2000.

[9] Mozilla.org, "Introduction to a XUL (XML-based User Interface Language) Document," available at http://www.mozilla.org/xpfe/xptoolkit/xulintro.html

[10] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., Shuster, J.E., "UIML: An Appliance-Independent XML User Interface Language," WWW8 Conf., May, 1999, available at http://www.harmonia.com/resources/papers/

[11] Sumanth Lingam, "UIML for Voice Interfaces," UIML Europe 2001 Conf., Mar., 2001, available at http://www.harmonia.com/resources/papers/

[12] Marc Abrams, "Device-Independent Authoring with UIML," W3C Workshop on Web Device Independent Authoring, Oct, 2000, available at http://www.harmonia.com/resources/papers/

[13] Andy Bower, Blair McGlashan, "Twisting the triad," tutorial paper for ESUG 2000, available at http://www.object-arts.com/Papers/

[14] Mozilla.org, http://mozilla.org/

[15] Krasner G., Pope S., "A cookbook for using the MVC user interface paradigm in Smalltalk". in *JOOP*, 1(3), pp.26-49, 1988.

최 종 명

1992년 숭실대학교 전자계산학과 학사. 1996년 숭실대학교 전자계산학과 석사. 1997년 ~ 현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 시각 프로그래밍, 멀티패러다임 시스템, XML

신 경 희

1985년 인하대학교 전자계산학과 학사. 1995년 숭실대학교 전자계산학과 석사. 현재 숭실대학교 컴퓨터학과 박사과정 수료. 관심분야는 프로그래밍 언어, 프로그래밍 환경, 시각 프로그래밍 언어, XML

유 재 우

1976년 학사 숭실대학교 전자계산학과. 1985년 박사 한국과학기술원 전산학과. 1983년 ~ 현재 숭실대학교 컴퓨터학부 교수. 1986년 ~ 1987년, 1996년 ~ 1997년, 코넬대학교, 피츠버그대학교 객원교수. 1999년 ~ 2000년 한국정보과학회 프로그래밍언어 연구회 위원장. 관심분야는 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용