그리드 포탈을 위한 객체 기반의 웹 사용자 인터페이스

(Object-oriented Web User-Interface Model for the Grid Portal)

고 윤 석 † 황 선 태 ††
(Yoonseok Ko) (Suntae Hwang)

요 약 그리드 포탈 서비스는 기존의 어플리케이션 기반 그리드 서비스가 갖는 시간적 공간적 제약을 극복하여 보다 유연한 연구 환경을 제공한다. 그러나 사용자 인터페이스와 같은 사용자 요구 사항은 개발 기간 동안 불분명하기도 하고 자주 변하기도 한다. 그러므로 그리드 서비스를 위한 사용자 인터페이스를 만들고 유지 관리하는 비용을 줄일 수 있는 방안이 필요하다. 본 논문에서는 유저 인터페이스 개발이 보다 쉽고 코드의 재사용률을 증가시킬 수 있는 객체 기반의 컨트롤 모델을 제안한다. 이 모델에서는 프레젠테이션 페이지의 구조를 보다 명확히 기술할 수 있고 액션들을 구현하는 것이 보다 쉽고 간단하다.

키워드 : GUI 프로그래밍, 그리드스피어, 객체지향 프로그래밍, 그리드 포탈, 사용자 인터페이스

Abstract Grid portal services provide more flexible research environment by overcoming time and space limit of existing application-based grid services. However user's requirements such as user interfaces are not clear during development cycle and changed frequently. Therefore it is necessary to reduce the cost for creating and maintaining user interfaces of grid services. In this paper, we suggest an object-oriented user control model which allows easier development of user interfaces and increases code reusability by abstracting objects from presentation layer of web. In this model, structure of presentation pages can be described more clearly and implementation of actions is simple and easy.

Key words : GUI Programming, GridSphere, Object-oriented Programming, Grid Portal, User-interface

1. 서 론

최근 그리드 서비스를 제공하기 위한 방법으로써 그리드 포탈 서비스가 대두되었다. 그리드 포탈 서비스는 원거리에 있는 여러 학자가 이기종의 시스템을 사용하여 동일한 자원을 사용하기에 적합한 환경이며 사용자 인터페이스의 구현이 쉽다는 장점을 갖고 있다. 이러한 추세에 따라 웹 포탈 어플리케이션의 개발이 그리드 서비스를 제공하는 입장에서 매우 중요한 이슈가 되고 있다.

최근에 그리드 포탈 개발을 위해 많이 사용되는 프레임워크로써 그리드스피어(GridSphere)[1]가 있다. 그리드스피어는 작은 페이지 조각의 단위인 포틀릿 단위로 서비스를 생성함으로써 필요한 서비스를 원하는 형태로 페이지 안에 배치하여 직접 화면을 구성할 수 있다는 장점이 있다. 또한 각각의 포틀릿[2]은 독립적이므로 배포가 쉽고 동일한 프레임워크 상에서 동작하므로 인터페이스가 일관적이다.

하지만 기존의 웹 모델과 마찬가지로 모든 이벤트에 대한 결과로써 매번 화면을 새로 받아서 처리해야 한다. 그러므로 일반 어플리케이션에 비해 느리고 불편하다는 점을 감수해야만 한다. 또한 그리드 포탈 서비스 특성상 잦은 사용자 인터페이스의 변경이 요구되는데, 그리드스피어는 모든 이벤트 처리를 기존의 웹 모델과 동일한 폼 기반의 요청방식을 사용하고 있다. 그러므로 인터페이스 변경에 따라 이벤트 처리 코드 역시 변경되어야만 한다. 이는 유지비용의 상승을 야기한다[3].

한편 최근 웹 어플리케이션 개발에서 폼 기반의 요청

· 본 논문은 건설교통부 "분산연구형 건설연구인프라 구축 사업 추진 연구단"의 지원을 받아 수행된 연구입니다.
† 학생회원 : 국민대학교 전산과학과
mir597@kookmin.ac.kr
†† 종신회원 : 국민대학교 전산과학과 교수
sthwang@kookmin.ac.kr
(Corresponding author임)
논문접수 : 2006년 8월 25일
심사완료 : 2006년 9월 28일

을 통한 이벤트 처리 방식의 단점을 보완하기 위한 방안으로써 아약스 패턴이 이슈가 되고 있다[4]. 이는 요청 및 응답을 비동기식으로 처리함으로써 실제 필요한 데이타만 자바스크립트로 직접 주고받는 방식이다.

하지만 이러한 방식은 일정 이상의 프로그래밍 기술이 요구되며 복잡한 스크립트 코드가 사용된다. 그러므로 기존의 서비스에 쉽게 적용하기 힘들다는 문제점을 안고 있다.

본 논문에서는 인터페이스 작성 코드의 재사용성을 높여 잦은 인터페이스 변경에 따른 개발비용을 감소시키기 위하여 객체 기반의 웹 사용자 인터페이스 모델(OWUM: Object-oriented Web User-interface Model)을 제시한다. 본 모델은 기존 웹의 이벤트 처리방식의 단점을 보완하기 위하여 아약스 패턴을 사용하였다. 또한 모든 아약스 패턴 관련 코드를 개발 프레임워크에 포함시켜 추상화함으로써 구현 코드의 난이도를 낮추는 방안을 모색하였다.

본 논문의 구성은 다음과 같다. 2장에서는 객체 기반의 웹 개발 프레임워크 모델을 제안한다. 3장에서는 이러한 모델을 구현하기 위한 프레임워크로써 그리드스피어를 선택하여 그 위에서 객체 기반의 웹 인터페이스 모델을 적용 및 구현하는 것을 사례 연구로써 제시한다. 4장에서는 관련 연구를 소개하고 5장에서는 기존 모델과의 비교를 통하여 평가한다. 6장에서는 본 논문의 결론 및 향후 연구를 설명한다.

2. 객체 기반의 웹 사용자 인터페이스 모델

본 논문에서 제시하는 OWUM은 그림 1과 같은 계층으로 구성된다. 다음은 각 레이어의 주요 개념이다.

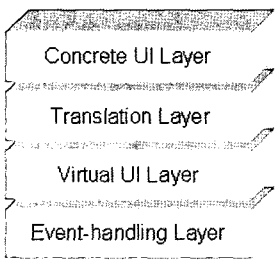

그림 1 OWUM의 계층도

- 실제 뷰 레이어(Concrete UI Layer): HTML, CSS[5] 등의 메타언어로 정의된 실제 컨트롤이 존재하는 영역
- 가상 뷰 레이어(Virtual UI Layer): 실제 뷰 레이어에 존재하는 각각의 컨트롤에 대한 가상 컨트롤 객체가 존재하는 영역
- 변환 레이어(Translation Layer): 실제 뷰 레이어와 가상 뷰 레이어 사이에 존재하며 양 측 컨트롤의 데이타를 동일하게 일치시켜주는 역할을 하는 영역
- 이벤트 처리 레이어(Event-handling Layer): 실제 뷰 레이어에서 발생된 이벤트의 처리를 담당하는 영역

2.1 실제 뷰 레이어

실제 뷰 레이어는 페이지의 첫 번째 요청에 의해 생성된다. 첫 번째 요청은 일반적인 웹 환경과 동일하게 해당 URL을 서버에 요청함으로써 발생한다. 페이지는 JSP 커스텀 태그 라이브러리를 통해 생성 및 초기화된다[6]. 이후에는 이벤트 처리 요청을 통하여 페이지가 갱신된다.

2.2 가상 뷰 레이어

실제 뷰 레이어에 존재하는 모든 사용자 컨트롤에 대한 가상 객체로 이루어진다. 각각의 컨트롤은 태그 상에 선언된 id 값으로 실제 뷰 레이어의 컨트롤과 사상한다.

가상 객체는 처리하고자 하는 이벤트에 대한 실질적인 구현을 담고 있다.

2.3 변환 레이어

실제 뷰 레이어와 가상 뷰 레이어 사이에서 서로 다른 구현을 통하여 이루어진 양쪽 컨트롤의 속성 값을 동일하게 유지하는 역할을 한다.

실제 뷰 레이어와 가상 뷰 레이어는 인터넷을 기반으로 서버와 클라이언트 구조를 갖고 있으므로 변환 레이어는 각 모델의 속성 데이타를 주고받는 역할을 포함한다.

2.4 이벤트 처리 레이어

이벤트 처리에 대한 정보는 가상 컨트롤 객체에 직접 구현한다. 각 이벤트 처리에 대한 코드는 자바 코드로 작성되며 실제 뷰 레이어를 추상화 한 가상 컨트롤 객체를 다룬다. 그러므로 실제 뷰 레이어의 실질적인 구현에 관계없이 가상 뷰 레이어의 모델을 컨트롤 하는 것만으로 사용자 인터페이스를 다룬다.

2.5 사용자 컨트롤의 정의

사용자 컨트롤은 컨테이너 타입과 컨트롤 타입으로 구분할 수 있다.

컨테이너 타입은 다른 사용자 컨트롤을 담는 역할을 하며 컨트롤의 값이 변경된 경우 화면 레이아웃 상에서 컨트롤의 위치가 변경된다. 또한 값이 추가되고 삭제됨에 따라 화면 레이아웃이 변경될 수 있다. 컨테이너 타입의 컨트롤이 갖는 값은 포함하고 있는 사용자 컨트롤의 id 값이며 순서 있는 리스트로써 표현된다.

컨트롤 타입은 이벤트를 발생 시킬 수 있다. 컨트롤의 값이 변경된 경우 컨트롤이 표현하는 값이 변경되며 레이아웃이 변경되지는 않는다.

각각의 사용자 컨트롤을 구현하기 위해서는 여러 가지로 분류된 코드를 직접 작성해야 한다. 사용자 컨트롤의 구현은 클라이언트 스크립트를 포함하므로 조금 까

다룹다. 그러므로 기본위젯에 대한 컨트롤을 미리 정의하여 제공해야 한다. 기본위젯의 종류에는 컨테이너 타입으로써 Window, Table 등이 제공되어야 하며 컨트롤 타입으로써는 Label, Image, TextBox, TextView, RadioButton, CheckBox, Button 등이 제공되어야 한다.

2.6 OWUM의 이벤트 처리 Lifecycle

이벤트가 발생하면 실제 뷰 레이어는 변환 레이어에 이벤트 처리를 요청한다. 변환 레이어는 실제 뷰에 존재하는 모든 컨트롤의 속성을 추출하고 발생된 이벤트 정보와 합쳐 서버로 전송한다. 서버에서는 각 컨트롤의 속성을 가상 뷰 레이어에 사상된 가상 컨트롤 객체에 삽입하고 발생된 이벤트 정보를 해당 컨트롤에 넘겨준다.

이벤트가 처리되고 나면 반대로 가상 컨트롤 객체의 모든 속성 데이타를 추출하여 실제 뷰 레이어로 전송한다. 이 때 데이타 전송은 HTTP의 요청/응답 메커니즘에 따라 응답으로 전송된다.

클라이언트는 미리 정의된 Callback Handler가 호출됨으로써 응답을 처리한다. 서버에서 전송된 컨트롤 객체의 모든 속성 데이타는 다시 실제 뷰 레이어의 컨트롤 객체에 삽입됨으로써 Lifecycle을 마무리 한다.

그림 2는 이러한 Lifecycle을 포함한 OWUM의 구현 계층도이다. 전체 계층구조는 MVC(Model View Controller)패턴의 Model 2 아키텍처에 기반을 둔다[7]. 아약스 패턴을 사용하여 서버와 데이타를 주고받기 위하여 RequestGenerator와 ResponseReflector가 사용된다. ObjectConvertor는 실제로 전송 데이타를 직접 받지 않으며 서블릿 기반의 데이타 처리 레이어가 필요하다. 하지만 기존의 개발 프레임워크에는 이러한 레이어가 이미 존재하므로 ObjectConvertor는 그 위에서 동작하는 것으로 구현될 수 있다.

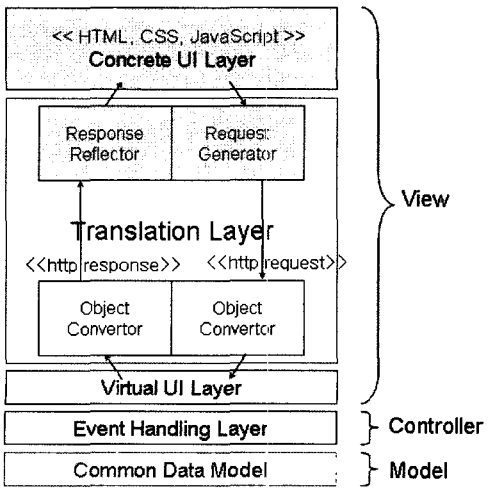

그림 2 OWUM의 구현 계층도

3. 구현 사례

본 논문에서 제안한 모델에 따른 사례로써 그리드스피어 2.x를 기반으로 OWUM을 구현하였다. 그리드스피어는 구조상 한 화면에 여러 개의 포틀릿이 존재한다. 그러므로 한 번의 요청은 이벤트 처리가 필요한 포틀릿으로 보내지는 액션 요청과 페이지에 속한 모든 포틀릿에 보내지는 페이지 생성 요청으로 분류할 수 있다.

본 논문에서는 기본 Life Cycle 외에 추가적인 Life Cycle을 구현하여 특정 값이 요청에 포함된 경우 새로 정의된 Life Cycle에 의하여 액션 요청이 처리되고 액션이 실행된 포틀릿만 페이지 생성 요청을 처리하도록 구현하였다.

3.1 그리드스피어 기반의 프레임워크 구현

3.1.1 사용자 컨트롤의 정의

사용자 컨트롤의 정의는 실제 뷰 레이어에서의 컨트롤 정의와 가상 뷰 레이어에서의 컨트롤 정의로 나뉜다.

첫째로 실제 뷰 레이어의 컨트롤 정의는 다음과 같다. JSP 커스텀 태그를 구현함으로써 컨트롤의 XHTML[8] 구조를 정의한다. 정의된 구조에 따라 XHTML 컨트롤의 값을 추출 및 삽입하기 위해서 DOM(Document Object Model)을 사용하는 JavaScript코드를 작성한다.

둘째로 가상 뷰 레이어의 컨트롤 정의는 다음과 같다. 커스텀 태그로 구현된 컨트롤과 동일한 속성 구조를 갖는 가상 컨트롤 객체를 선언한다. 가상 컨트롤 객체 역시 속성 값을 추출 및 삽입하기 위한 기본적인 Java코드를 작성한다.

커스텀 태그를 사용하여 선언하므로 하나의 컨트롤은 여러 개의 XHTML 객체로써 표현될 수 있으며, 반드시 well-formed XHTML이어야 한다. 각 컨트롤 객체의 값을 일관적으로 추출 및 삽입하는 코드를 작성할 수 있다면 어떠한 형태로든 구현될 수 있다.

3.1.2 변환레이어의 구현

변환레이어는 서버 코드와 클라이언트 코드로 분류된다. 실제로 변환레이어를 구현하기 위해서는 서버와 클라이언트 사이의 데이타 전송까지 구현해야 하며 이 부분은 클라이언트의 자바스크립트에서 아약스 패턴을 구현함으로써 처리한다.

변환 레이어는 RequestGenerator, ResponseReflector, ObjectConvertor로 구현된다. ObjectConvertor는 실제로 데이타를 직접 받지 않고 그리드스피어를 통해 전달받는다.

RequestGenerator는 각각의 컨트롤별로 미리 작성된 상태 추출 JavaScript코드를 호출한다. 모든 컨트롤의 상태 값은 하나의 문자열로 합쳐져서 서버로 전송한다. 서버에서는 문자열을 다시 컨트롤 별로 분리하여 각각

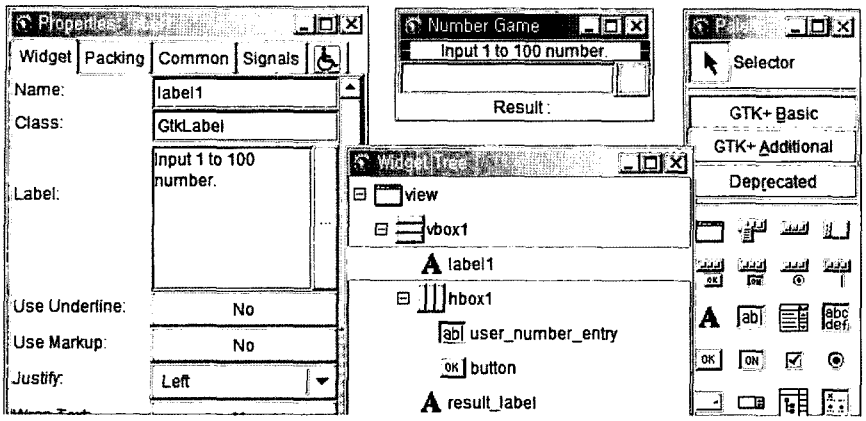

그림 3 윈도우즈용 글레이드의 사용자 인터페이스

의 컨트롤에 삽입함으로써 클라이언트에서의 객체 상태를 그대로 복원한다.

아약스 패턴을 이용한 비동기식 요청은 Callback Handler에 의해 응답을 받을 수 있다. 받은 응답은 Response-Reflector로 전달되어 다시 각각의 실제 객체에 삽입된다.

ObjectConvertor는 미리 작성된 상태 삽입 코드를 객체 별로 호출하여 각각의 컨트롤 상태 값을 객체에 입력함으로써 현재 클라이언트 브라우저에 HTML로 이루어져 있는 모든 사용자 컨트롤 객체를 서버의 가상 컨트롤 객체로 만들어낸다.

또한 이벤트 혹은 서버 액션이 수행된 후에는 서버 상의 가상 컨트롤 객체의 값을 다시 추출하여 클라이언트로 보낸다.

3.1.3 가상 뷰 레이어의 구현

가상 뷰 레이어는 자바로 구현된 가상 컨트롤 객체로 이루어져 있다.

가상 뷰 레이어는 변환 레이어, 컨트롤 레이어와의 인터페이스를 갖고 있으며 현재 클라이언트의 사용자 컨트롤의 상태를 그대로 반영하기 위한 레이어이다.

가상 뷰 레이어를 구현하기 위해서는 JSP 커스텀 태그에서 사용된 모든 객체를 해당 id 값에 따라 사용할 수 있도록 선언한다. 선언된 객체의 속성은 실행 Lifecycle에서 ObjectConvertor가 삽입한다.

3.1.4 레이아웃 및 골격 코드 생성

본 구현에서는 레이아웃 설계 및 변경을 쉽게 하고 이에 따른 골격 코드를 자동으로 생성하기 위해 XML[9] 변환기를 설계하였다.

레이아웃 XML을 생성하기 위해서 글레이드(Glade) 툴[10]을 사용하였다. 글레이드는 GTK라이브러리를 사용하는 어플리케이션의 UI 생성 툴로써 화면 레이아웃을 정의하고 위젯을 배치하면 그 결과로써 XML문서를 생성한다. 이 결과물은 커스텀 태그로 이루어진 JSP문서 및 골격 코드를 생성하기에 적합한 정보를 담고 있으므로 이를 활용하였다.

XML 변환기는 글레이드의 결과인 XML 문서를 입력으로 받아 이에 해당하는 JSP문서와 골격 코드를 생성한다. 그림 4는 글레이드의 결과물인 XML 문서이며 그림 5와 6은 변환기를 통하여 자동으로 생성한 JSP문서 및 자바 골격 코드이다.

3.2 예제: 간단한 숫자 맞추기 게임

구현된 프레임워크 상에서 간단한 숫자 맞추기 게임을 개발해 보았다. 그림 7은 본 구현에서 개발하고자 하는 숫자 게임의 사용자 인터페이스이다. 이를 쉽게 생성하기 위하여 글레이드를 사용하였으며 이를 XML 변환기를 통해 JSP문서와 골격 코드를 생성하였다.

생성된 골격코드에는 숫자게임을 구현하기 위하여 그림 8과 같이 간단한 이벤트 처리 코드를 삽입한다. 이러한 코드는 코드 생성기가 생성한 골격 코드에 약간의 코드를 추가하여 웹 어플리케이션의 프레젠테이션 계층을 완성할 수 있음을 보여주고 있다.

그림 9는 완성된 모델에 대한 UML 클래스 다이어그램으로써 모델의 구조가 명확한 다이어그램으로 표현 가능하다는 것을 보여주고 있다.

4. 관련연구

최근 많이 사용되는 웹 인터페이스 개발 모델 으로는 JSP의 JSTL(Java Standard Template Library) 같은 표준 라이브러리와 커스텀 태그 라이브러리 모델이 있다. 커스텀 태그 라이브러리는 데이타 모델을 HTML모델과 결합시켜 동적으로 사용자 인터페이스를 생성한다. 이러한 모델은 사용자 인터페이스를 객체 단위로 생성하므로 동적인 페이지를 쉽게 생성할 수 있다는 장점이

```
<?xml version="1.0"?>
<glade-interface>
<widget class="GtkWindow" id="view">
<property name="title">Number Game</property>
<child>
    <widget class="GtkVBox" id="vbox1">
    <child>
        <widget class="GtkLabel" id="label1">
        <property name="label">Input 1 to 100
number.</property>
        </widget>
    </child>
    <child>
        <widget class="GtkHBox" id="hbox1">
        <child>
            <widget class='GtkEntry'
                id="user_number_entry">
            <property name="text"/>
            </widget>
        </child>
        <child>
            <widget class='GtkButton'
                id="button">
            <signal name="clicked"/>
            </widget>
        </child>
        </widget>
    </child>
    <child>
        <widget class="GtkLabel" id="result_label">
        <property name="label">Result : </property>
        </widget>
    </child>
    </widget>
</child>
</widget>
</glade-interface>
```

그림 4 글레이드를 통해 생성된 XML

```
<%@ taglib uri="/owumUI" prefix="ui" %>

<ui:window id='view'>
    <ui:vbox id='vbox1'>
        <ui:label id='label1'>Input 1 to 100 number.
</ui:label>
        <ui:hbox id='hbox1'>
            <ui:textbox id='user_number_entry'/>
            <ui:button id='button' clicked='on'/>
        </ui:hbox>
        <ui:label id='result_label'/>
    </ui:vbox>
</ui:form>
```

그림 5 자동 생성된 JSP 문서

```
public class wndView extends Window
{
    public Label label1 = new Label();
    public Label result_label = new Label();
    public Hbox hbox1 = new Hbox();
    public Vbox vbox1 = new Vbox();
    public Textbox user_number_entry
                    = new Textbox();
    public Button button = new Button();

    public void init()
    {
        this.jspLocation="view.jsp";
        button.addActionListener(new ActionListener()
        {
        public void actionPerformed(ActionEvent e)
        {
            // TODO
        }
        });
    }
}
```

그림 6 자동 생성된 골격 코드

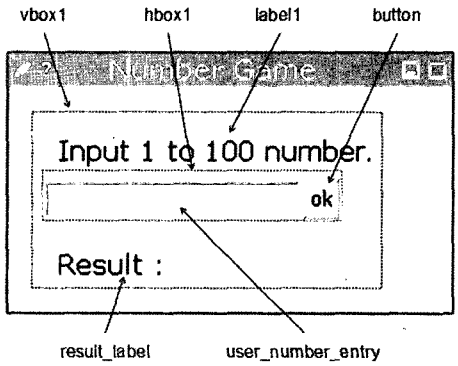

그림 7 숫자게임의 사용자 인터페이스

```
public void actionPerformed(ActionEvent e)
{
    int val =
 Integer.parseInt(user_number_entry.getValue());
    if (val > number)
        result_label.setValue("high");
    else if (val < number)
        result_label.setValue("low");
    else
        result_label.setValue("right!");
}
```

그림 8 OWUM이 적용된 그리드스피어 숫자게임 코드

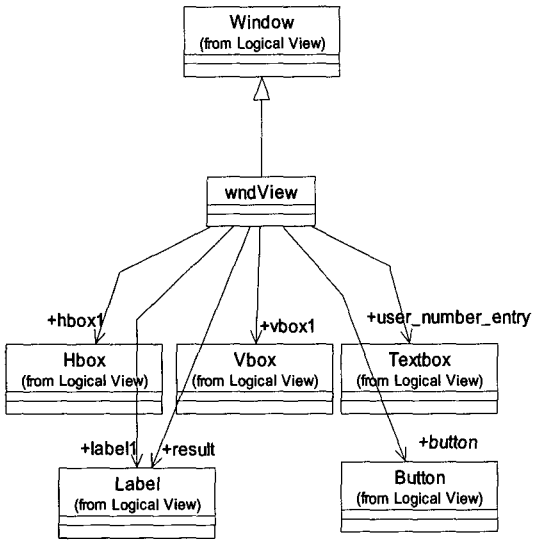

그림 9 숫자 게임의 UI 클래스 다이어그램

있다. 하지만 커스텀태그를 사용한 기존의 구현은 동적으로 모델의 배치를 변화시킬 수 없으며 출력된 이후에는 모델을 직접 동적으로 다루지 못한다는 단점이 있다.

아약스 패턴을 쉽게 다루기 위한 모델로써 DWR(Direct Web Remoting)[11]이 있다. 이는 간단하게 클라이언트의 자바스크립트에서 직접 서버의 메소드를 호출할 수 있으며 데이타를 사용할 수 있다는 장점이 있다.

이와 유사한 형태로 ASP.NET[12]의 비공식적인 AJAX지원 형태로써 AJAX.NET[13]등의 프로젝트가 있다. 이는 서버 코드에 간단한 메타 표기를 추가함으로써 DWR과 유사하게 클라이언트 자바스크립트에서 서버의 메소드를 직접 호출할 수 있는 메커니즘을 제공하고 있다.

DWR과 AJAX.NET은 클라이언트를 하나의 독립적인 어플리케이션으로써 생성하고 서버와 데이타를 주고받기 위한 메커니즘으로 아약스를 사용하고 있다. 즉 아약스를 사용하기 위한 환경으로써 클라이언트 자바스크립트 환경을 위주로 하고 있다. 이는 클라이언트와 서버의 개발을 분리함으로써 협업에 유리한 환경을 구축할 수 있다. 반면 클라이언트 코드가 복잡해지고 하나의 사용자 인터페이스를 구성하기 위해서 서버 측과 클라이언트 측으로 나뉘어 각각 자바와 자바스크립트로 코드가 생성되어야 한다. 그러므로 설계 단계에서 서버와 클라이언트 코드 사이의 인터페이스가 명확해야 하며, 이에 관련한 오류는 디버깅하고 수정하기 까다롭다. 그러므로 개발과 유지보수가 어려워진다는 단점을 야기한다.

UIML(User Interface Markup Language)은 사용자 인터페이스를 기술하기 위한 표준 메타언어이다[14]. 이는 본 논문에서 구현하고 있는 글레이드의 결과 XML 문서와 흡사한 형태이며 더욱 상세하고 명확한 구조를 갖고 있다. 하지만 이러한 메타언어를 통하여 XML 문서를 생성하는 위지윅 툴은 아직 개발 중이다. 그러므로 본 연구에서는 이미 개발이 완료된 글레이드를 활용하였다.

XUL(XML User Interface Language)은 어플리케이션 기반의 사용자 인터페이스를 생성하기 위한 메타언어이다[15]. 이는 XML 문서를 통하여 사용자 컨트롤을 배치하고 자바스크립트를 사용하여 액션을 정의한다. XUL은 HTML에 비해 강력한 컨트롤을 지원하며 어플리케이션의 사용자 인터페이스를 쉽게 생성할 수 있다는 장점을 갖는다. 이는 본 논문에서 구현하고 있는 글레이드 기반의 사용자 인터페이스 설계 및 자동변환기를 통한 골격코드 생성과 유사한 형태로써 참고할 수 있다.

5. 비교 및 평가

본 연구의 결과인 객체 기반의 사용자 인터페이스 모델을 사용하는 그리드스피어 프레임워크와 기존의 그리드스피어 프레임워크의 기능과 비교하였다.

그림 8과 10은 각각 OWUM이 적용된 모델과 기존 그리드스피어에서 숫자 게임을 구현하기 위한 이벤트를 정의하는 코드이다. 객체 모델이 적용된 코드는 훨씬 직관적이고 간편하게 웹 어플리케이션의 프레젠테이션 계층을 완성할 수 있음을 보여주고 있다.

OWUM은 실제 뷰 레이어와 가상 뷰 레이어 간의 속

```
public void valueCheck(ActionFormEvent event)
                        throws PortletException
{
    TextFieldBean entry
      = event.getTextFieldBean("user_number_entry");
    String val = entry.getValue();
    ActionResponse res = event.getActionResponse();
    int value=-1;

    if (val != null)
        value = Integer.parseInt(val);
    if (value == -1)
        res.setRenderParameter("result", "");
    else if (number > value)
        res.setRenderParameter("result", "low");
    else if (number < value)
        res.setRenderParameter("result", "high");
    else
        res.setRenderParameter("result", "ok! Good!");
    setNextState(event.getActionRequest(),
                 DEFAULT_VIEW_PAGE);
}
```

그림 10 기존의 그리드스피어 숫자게임 코드

성을 일치시키기 위한 비용을 요구한다. 특히 클라이언트는 자바스크립트를 사용하므로 컴퓨터 사양에 따라 그 비용이 민감한 문제가 될 수 있다. 그림 11은 펜티엄 3 1.12Ghz의 클라이언트에서 페이지 당 컨트롤의 수에 따른 각각의 이벤트 처리에 필요한 시간을 그래프로 표현하고 있다. 컨트롤이 없을 때의 응답 시간은 320ms로써 이는 네트워크 고유의 속도와 서버의 처리속도를 의미한다. 이는 로컬서버에서 테스트 하였으며 숫자 게임을 위한 페이지가 작으므로 작은 수치가 측정되었다.

한 페이지 내에 컨트롤이 200개가량 존재할 때의 컨트롤 처리 시간은 전체 처리 시간의 약 41%를 차지한다. Microsoft의 WAS(Web Application Stress) 툴을 사용하여 www.google.com등의 포탈 메인 페이지 요청 시간을 테스트 한 결과 평균 TTLB는 579ms이다. 이를 일반 네트워크의 네트워크 및 서버의 처리 시간이라 가정하면 일반 네트워크를 기준으로 할 경우 컨트롤 처리에 필요한 시간은 전체 요청처리 시간의 약 28%정도를 차지함을 알 수 있다. 또한 한 가상 윈도우 내에 수백 개의 컨트롤이 들어갈 필요가 없다는 것을 감안한다면 요청처리에 필요한 컴퓨팅 비용 및 시간은 더욱 줄어들 것이다.

그림 12는 이벤트 처리 요청에 요구되는 데이타 전송량을 나타내고 있다. 본 요청 및 응답에는 이미지 등의 비교적 많은 용량을 필요로 하는 데이타가 포함되지 않는다. 그러므로 순수하게 요청을 처리하기 위한 데이타의 용량이라 할 수 있다. 클라이언트는 요청을 처리하기 위해 데이타를 생성하고 응답을 처리하기 위해 데이타를 다시 해석하는 과정을 필요로 한다. 그러므로 주고받는 데이타의 용량은 클라이언트에서 요청을 처리하기 위해 필요한 컴퓨팅 비용 및 시간이라 할 수 있다.

기존의 그리드스피어 프레임워크가 사용하고 있는 폼 기반의 요청 처리방식은 보내는 데이타양이 비교적 적은 편이다. 반면 응답으로써 페이지 전체를 받기 때문에 데이타양이 비교적 많다. 또한 기존에 해석한 HTML문서의 DOM을 제거하고 새로 받은 데이타를 다시 해석하여 DOM을 생성하므로 화면 리프레시가 발생하고 데이타 처리 시간이 오래 걸린다는 단점을 갖고 있다.

반면 아약스를 기반으로 하는 그리드스피어는 주고받는 데이타가 매우 적다. 이는 요청 처리에 필요한 비용이 적다는 것을 의미한다. 또한 비동기식 요청을 사용하므로 이벤트 처리에 관련된 동작은 내부에서 진행되며 기존의 DOM을 그대로 둔 채 데이타를 주고받은 후 응답 데이타의 결과에 따라 DOM을 수정하여 페이지를 변경하는 구조를 갖고 있다. 그러므로 화면 리프레시는 발생하지 않으며 효과적으로 사용자 인터페이스를 다룰 수 있다는 장점이 있다. 하지만 이러한 구현을 위해서는 각각의 이벤트 처리를 위한 서버 코드와 클라이언트 코드가 각각의 언어로 동시에 작성되어야 한다. 그러므로 작성이 까다롭고 개발 및 유지보수에 필요한 코드가 분산됨으로써 관리 및 수정이 까다로워진다는 단점이 있다.

OWUM을 기반으로 하는 그리드스피어는 역시 아약스를 기반으로 하고 있다. 그러나 아약스만 사용하여 구현한 모델에 비해서는 비교적 많은 데이타를 주고받는다. 하지만 아약스에 대한 구현을 프레임워크에서 추상화하고 있으므로 각각의 컨트롤 배치와 이에 대한 이벤트 처리 코드를 서버에서만 작성함으로써 구현을 완료할 수 있다는 장점이 있다. 또한 OWUM의 사용자 컨트롤은 객체 모델을 기반으로 하고 있다. 그러므로 설계 단계에서도 더욱 명확한 UML 다이어그램으로 표현이 가능하다는 이점이 있으며 기존의 그리드스피어 모델에

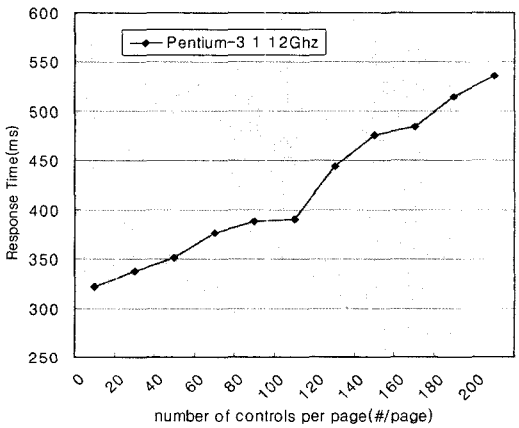

그림 11 컨트롤 수에 따른 응답 처리시간

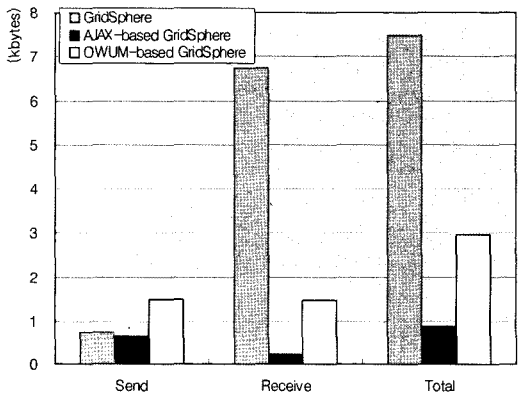

그림 12 각 모델을 적용한 그리드스피어에서 요청처리에 요구되는 데이타 전송량

비해서도 작성되는 코드가 간단하므로 개발 및 유지보수가 용이하다는 장점이 있다.

6. 결론 및 향후 연구

본 연구에서는 일반 어플리케이션 환경에서 사용되던 객체 기반의 사용자 컨트롤 모델이 컨트롤 배치 등의 인터페이스 변화에 유연하며 좀 더 명확한 모델로써 표현이 가능하다는 점에 착안하여 웹 환경에서 동일한 모델을 적용할 수 있는 기법을 조사하였으며 이에 대한 모델을 설계하고 구현하였다.

OWUM은 각 사용자 인터페이스를 가상 윈도우 클래스로 선언하고 각 컨트롤을 클래스 객체로써 표현한다. 이는 UML의 클래스 다이어그램을 사용하여 각각의 윈도우와 컨트롤의 사용관계를 표현함으로써 개념설계단계에서 좀 더 명확한 클래스 관계도가 도출 가능하다는 장점이 있다. 또한 이후 과정에서도 명확한 다이어그램으로써 각 윈도우의 기능을 유추할 수 있으므로 인터페이스 변경이나 확장에 대한 유지보수 비용을 감소시킬 수 있다. 또한 기본 골격코드가 단순하기 때문에 XML로 화면 모델을 설계하고 자동 생성기를 통하여 골격코드 및 컨트롤 배치에 대한 코드를 생성할 수 있으므로 개발자가 적은 학습량으로 웹 어플리케이션을 개발할 수 있다는 이점이 있다. 따라서 개발자는 웹 어플리케이션을 보다 빠르고 효율적으로 개발할 수 있는 장점을 가진다.

본 연구는 모든 이벤트에 대한 처리를 서버에서 해야만 한다는 제약 조건을 갖는다. 클라이언트 환경에서 자바스크립트를 통해 간단한 이벤트 처리가 가능하다는 점과 마우스 이동과 같은 빠른 이벤트 처리가 필요한 부분은 반드시 자바스크립트를 통해 이벤트 처리가 이루어져야 한다는 점에서 한계점을 가지고 있다.

따라서 향후 연구 과제로는 클라이언트 액션에 대한 정의를 내리고 요청 및 응답에 요구되는 비용을 줄이는 방안을 모색하고자 한다. 또한 비즈니스 로직 레이어와의 연동을 통하여 데이타 모델을 직접 연결하여 좀 더 자동화된 모델을 정의하는 등 보다 심층적인 연구를 진행하고자 한다.

참 고 문 헌

[1] Jason Novotny, Michael Russell, Oliver Wehrens, "GridSphere: An Advanced Portal Framework," euromicro, pp.412-419, 30th EUROMICRO Conference(EUROMICRO'04), 2004.

[2] Java Community Process: JSR 168 Portlet Specification. Project Website, available at http://www.jcp.org/jsr/detail/168.jsp

[3] A. Saimi, T. Syomura, H. Suganuma, and I. Ishida, "Presentation Layer Framework of Web Application Systems with Server-side Java Technology," COMPSAC 2000, The Annual International, pp. 473-478, 2000.

[4] Paulson, L.D., "Building rich web applications with Ajax," Computer, vol.38, no.10, pp.14-17, Oct, 2005.

[5] World Wide Web Consortium, Cascading Style Sheets(CSS) Level 1 Specification, tech. report, available at http://www.w3c.org/TR/REC-CSS1

[6] Java Community Process: JSR 152 JavaServer Pages 2.0 Specification. Project Website, available at http://www.jcp.org/jsr/detail/152.jsp

[7] Steve Burbeck, "Application Programming in SmallTalk-80 : How to use Model View Controller (MVC)," available at http://st-www.cs.uiuc.edu/users/march/st-docs/mv.html. 1992.

[8] W3C, XHTML 1.0 Specification, tech. report, available at http://w3.org/TR/xhtml1/

[9] World Wide Web Consortium, Extensible Markup Language(XML) 1.0 Specification, tech. report, available at http://www.w3c.org/TR/REC-xml.

[10] Glade - a User Interface Builder for GTK+ and GNOME, Project Website, available at http://glade. gnome.org/

[11] DWR - Direct Web Remoting, Project Website, available at http://getahead.ltd.uk/dwr

[12] Microsoft ASP.NET. Project Website, available at http://asp.net/

[13] AJAX.NET Professional. Project Website, available at http://www.ajaxpro.info/

[14] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, Jonathan E. Shuster, "UIML: An Appliance-Independent XML User Interface Language," WWW8, Computer Networks, 1999.

[15] XML User Interface Language(XUL) 1.0 Specification, tech. report, available at http://www.mozilla.org/projects/xul/xul.html

고 윤 석

2006년 국민대학교 전산과학 석사 과정
2005년 국민대학교 컴퓨터학부(학사). 관심분야는 그리드 시스템, 소프트웨어공학

황 선 태

1985년 서울대학교 컴퓨터공학과(학사)
1987년 서울대학교 컴퓨터공학과(석사)
1996년 Manchester University (PhD)
1997년~ 국민대학교 컴퓨터학부 부교수
관심분야는 e-Science, 그리드시스템, PSE, 공개소프트웨어