Ajax기반 웹 응용을 위한 아키텍처 패턴 설계

준회원 김 황 만*, 종신회원 김 용 구*°

Design of an Architecture Pattern for Ajax-based Web Applications

Hwang-man Kim* *Associate Member*, Yong-Goo Kim*° *Lifelong Member*

요 약

복잡한 Ajax (Asynchronous JavaScript and XML) 기반 클라이언트 개발에 있어, 그 개발편의성을 증대시키고 방대한 코드의 유지보수성을 개선하기 위해, 본 논문에서는 MVC (Model-View-Client) 프레임워크 기술을 변형한 CVC (Communicator-View-Controller) 아키텍처 패턴을 제안한다. 제안된 CVC 아키텍처 패턴은 Ajax 기반 클라이언트가 공통적으로 가지게 되는 데이터 추출을 위한 비동기 통신 관련 코드를 Communicator 영역으로 분류함으로써, 그래픽 디자인 영역에 해당하는 View와, View 계층에 효과적으로 데이터를 갱신하는 Controller 영역으로 개발 코드를 구분한다. 이와 같은 구분을 통해 Ajax 기반 클라이언트 개발의 방법을 서술적으로 개념화시킴으로써, Ajax 관련 코드를 효과적으로 모듈화 하여 재사용하고 그래픽 디자인을 독립적으로 처리할 수 있도록 하여 Ajax 기반 웹 응용의 개발생산성 및 유지보수성을 획기적으로 높일 수 있도록 하였다.

Key Words : Ajax, user interface design, MVC, software architecture pattern, complex web client

ABSTRACT

In order to achieve the ease of development and to facilitate the maintenance of codes for complex Ajax (Asynchronous Java Script and XML)-based web clients, this paper proposes a CVC (Communicator-View-Controller) architecture pattern by modifying the well-known MVC (Model-View-Controller) framework. By composing the Communicator of codes for asynchronous data retrieval, which is common to Ajax-based clients, the proposed architecture pattern is able to cut out the graphic design related codes to constitute the View layer. Based on such declarative generalization of complex web-client codes, Ajax-related codes can be easily modularized and efficiently reused in development and maintenance stages, and graphics design can be done separately regardless of the other business logic related codes development, resulting in highly efficient development and maintenance of complex Ajax-based web clients.

Ⅰ. 서 론

1990년대 등장한 웹 기술은 네트워크의 보급 확산과 정보공유 및 응용개발의 편의성에 힘입어 폭발적인 성장을 거듭하였다. 오늘날 웹 응용은 개인적 정보공유로부터 기업의 자원 관리 및 전자정부에 이르기까지 현대인의 생활 전반에 매우 깊이 자리 잡고 있는 필수적인 부분이 되었다.

사용자에게 있어 이와 같은 웹 응용의 모든 기능은 사용자 인터페이스를 통해 보여 진다. 이러한 사용자 인터페이스는 웹의 종단 사용자들이 경험하는 대화형 워크플로우의 모든 부분에 연관되며, 개발과정에서 가장 많은 시간과 노력이 요구되는 영역이다. 한 통계자료에 따르면, 웹 응용 시스템 개발에 있어서 전체 개

* 한독미디어대학원대학교 뉴미디어학부 미디어통신연구실 (ygkim@kgit.ac.kr) (° : 교신저자)
논문번호 : KICS2011-05-229, 접수일자 : 2011년 5월 26일, 최종논문접수일자: 2011년 9월 15일

발 비용 중 유지보수에 관계된 비용은 80%가 사용자 인터페이스와 관련이 있다고 한다[1].

Ajax (Asynchronous Java Script and XML)는 클라이언트 영역에 사용되는 일련의 웹 개발 방법들을 지칭하는 용어로 이러한 사용자 인터페이스 개발을 위한 대표적 기술 중 하나이다[2]. Ajax를 통하여, 웹 응용들은 현재 페이지의 디스플레이나 동작에 방해받지 않고 비동기 방식으로 서버로부터 데이터를 검색하고 추출하는 것이 가능하다. Ajax 기술은 웹 사이트의 능률적이고 역동적인 사용자 인터페이스 개발을 위해 폭넓게 채택되고 있으며[3-4], e-mail 시스템[5], 메시지 링킹[6], 주식거래[7] 및 전자정부 시스템[8-9] 등의 다양한 웹 응용에 향상된 사용자 경험을 제공하기 위해 사용되고 있다. Ajax 기반 클라이언트는 이와 같은 다양한 웹 응용에서 고급 사용자 경험의 제공을 가능하게 하는 장점이 있지만, 방대한 분량의 코드로 이루어진 복잡한 클라이언트 개발에 있어 웹-개발의 편의성과 유지보수성 등이 낮아지는 단점이 있다.

이와 같은 웹-개발의 편의성과 유지보수성 향상을 제공할 수 있는 기술로 MVC (Model-View-Controller) 프레임워크를 들 수 있는데, 이 기술은 응용 로직을 사용자 인터페이스와 분리함으로써 구조적 디자인 복잡도를 감소시키고 이를 통해 유지보수성 향상을 제공한다[10]. MVC 모델은 Smalltalk80의 사용자 인터페이스 디자인에서 발전된 기술로[11], 다양한 웹 응용 개발에 보편적으로 적용되는 기술이다. Yan은 멀티 에이전트 기반의 사용자 인터페이스 디자인에 MVC 모델을 적용하였는데[12], 에이전트들의 세부 기능을 분류하여 Model과 Controller 영역으로 구분하고 그래픽 기능을 수행하는 에이전트를 View로 분류함으로써 멀티 에이전트 기반의 웹 응용 구현을 체계화 하였다. 하지만, 복잡한 사용자 인터페이스 코드 영역을, 일반적인 서버-클라이언트 기반의 MVC 모델에서와 같이, 단순히 View 영역으로 구분함으로써 개발 편의성과 유지보수성을 개선하는데 제한적일 뿐 아니라, 다소 범용 적이지 못한 에이전트 기반의 사용자 인터페이스 구현에만 제한적으로 사용될 수 있는 기술이어서 Ajax 기반 웹 응용에 적용이 불가능하다. 한편, Stocklein 등은 차세대 사용자 인터페이스의 개발을 위해 MVC 모델을 확장하여 새로운 아키텍처 패턴을 제안하였다[13]. 제안된 아키텍처는 차세대 사용자 인터페이스의 주요 요소인 다양한 센서 입력을 처리하기 위해 기존의 MVC 모델에 E (Environment) 차원을 도입한 MVCE 아키텍처로, 사용자 인터페이스 응용이 요구하는 센서 정보와 실제 센서에서 발생시키는 정보량의 차이를 효과적으로 처리함으로써 가상현실 또는 복합현실을 기반으로 하는 차세대 사용자 인터페이스의 구현에 효과적인 프레임워크를 제공하였다. 하지만, 이 아키텍처 역시 Ajax를 기반으로 하는 복잡한 범용 웹 응용의 사용자 인터페이스 (또는 클라이언트) 구현에 효과적으로 적용할 수 없다.

복잡한 Ajax 기반 클라이언트 개발에 있어, 그 개발편의성을 증대시키고 방대한 코드의 유지보수성을 개선하기 위해, 본 논문에서는 MVC 프레임워크 기술을 변형한 CVC (Communicator-View-Controller) 아키텍처 패턴을 제안한다. CVC 아키텍처는 Ajax 기반 클라이언트가 공통적으로 가지게 되는, 데이터 추출을 위한 비동기 통신 관련 코드를 Communicator 영역으로 분류함으로써, 그래픽 디자인 영역에 해당하는 View와, View 계층에 효과적으로 데이터를 갱신하는 Controller 영역으로 개발 코드를 분류한다. 이와 같은 구분을 통해, Ajax 기반 클라이언트 개발의 방법을 서술적으로 개념화시킴으로써, Ajax 관련 코드를 효과적으로 모듈화 하여 재사용하고 그래픽 디자인을 독립적으로 처리할 수 있도록 하여 Ajax 기반 웹 응용의 개발생산성 및 유지보수성을 획기적으로 높일 수 있도록 하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 일반적인 Ajax 기반 클라이언트의 구조적 특징을 살펴보고, 제 3장에서는 MVC 모델을 변형한 제안 아키텍처 패턴인 CVC 모델을 설명한다. 4장에서는 제안 아키텍처 패턴의 개발생산성과 유지보수성 개선효과를 검증하기 위해, 간단한 웹-기반 회원관리 시스템을 구현하고 CVC 모델을 적용하는 경우와 그렇지 않은 경우에 대한 비교 연구 결과를 보인다. 마지막으로, 5장에서 본 논문의 결론을 맺는다.

Ⅱ. Ajax 기반 클라이언트의 구조적 특징 및 문제점

웹 응용에서 Ajax를 활용하기 위해서는 반드시 웹 어플리케이션 서버가 (Web Application Server) 필요한데, 이는 Ajax 자체 기술만으로는 데이터 서비스를 위한 활용도가 떨어지고, 로컬 데이터 밖에 서비스 할 수 없는 기능적 제한이 크기 때문이다. 또한 웹 브라우저와 서버 간 비동기 통신이라는 기능 하나만으로도 Ajax의 역할은 충분하지만 이 비동기 통신의 대응 상대가 웹 어플리케이션 서버이기 때문이다.

현행 Ajax 아키텍처는, 이와 같은 웹 응용 서버로부터 데이터를 얻기 위해, 단순히 서버에 데이터 요청

을 수행하는 대신 보다 높은 데이터 처리 효율을 위해 서버 영역의 코드를 포함하는 서버 종속적 형태를 가지고 있다. 그림 1에 이러한 서버 종속적 현행 Ajax 웹 응용 구조를 도시하였다. 보다 상세히, J2EE Model2 구조로 현행 웹 응용의 형태를 분석해 보면, Ajax 페이지 (Web Browser)의 소스는 View 영역 안에 서버사이드 스크립트 언어 (JSP, ASP, PHP등)를 기반으로 하는 Scriptlet, Custom Tag 또는 EL(Expression Language) 등을 포함하고, 여기에 Ajax를 구동하기 위한 Javascript 가 더해져 있다. 따라서 View 영역의 소스코드 복잡도는 일반 JSP나 ASP 페이지보다 훨씬 높아지게 되고, 웹 응용을 웹 어플리케이션 서버와 서버기술에 종속적이게 만든다. 뿐만 아니라, 이러한 소스코드의 복잡도 증대는 소스의 가독성을 떨어뜨려 유지보수의 효율을 떨어뜨리고, 디자인영역과 개발영역의 구분을 모호하게 함으로써 개발생산성의 저하 및 업무담당자 간 분쟁을 유발시킬 수도 있다.

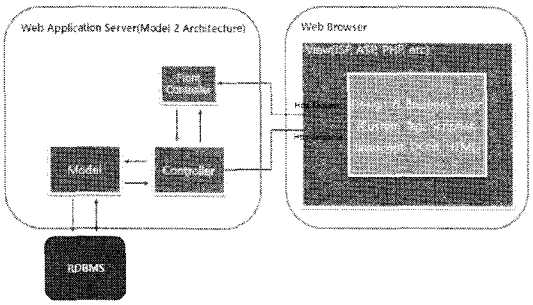


그림 1. Model 2 기반의 현행 Ajax 웹 응용 아키텍처
Fig. 1. Ajax architecture based on Model 2

Ⅲ. CVC 아키텍처 패턴

3.1 현행 Ajax기반의 웹 응용 아키텍처의 문제점 해결방안

표 1은 제 2장에서 설명했던 현행 Ajax기반 웹 응용 아키텍처의 문제점과 이를 해결하기 위한 본 논문의 개선방안을 정리한 것이다. 기존 J2EE기반의 웹 응용 개발에서 Model 1 아키텍처의 단점을 개선하기 위해 MVC 패턴 기반의 Model2 아키텍처 패턴[14]을 적용 했던 사례를 상기하여, Model 1 아키텍처에서의 문제점들과 매우 유사한 Ajax 기반 웹 응용의 문제점들도 기본적으로 MVC 패턴의 기본 정신을 적용함으로써 상당부분 해결 할 수 있을 것으로 기대된다.

표 1. 현행 Ajax기반 웹 응용 아키텍처의 문제점과 개선방안
Table 1. Problems of Ajax architecure for web applications and the proposed solutions

문제점	개선방안	문제점 유형
어플리케이션 모듈의 재사용성과 확장성이 떨어짐	각 역할간 기능을 Model, View, Controller 영역으로 구분하고 서로간의 독립성을 보장함으로써 소스코드의 재사용성을 높이고 테스트의 용이함을 제공	개발 생산성
웹 어플리케이션 서버와 서버기술에 종속적	클라이언트의 서버에 대한 종속성을 제거하기 위해 클라이언트를 서버관점에서의 View영역이 아닌 클라이언트 관점에서 서버를 Model 영역으로 정의	개발 생산성
소스코드의 복잡성이 높고 가독성이 떨어짐	Model, View, Controller 소스를 기능별로 각기 다른 파일로 저장하고, 해당 기능을 수행하는 파일을 import 함으로써 소스코드의 길이를 줄이고 가독성을 높임	유지 보수성
소스코드의 디자인영역과 개발영역의 구분도가 낮음	Controller에서 디자인된 오브젝트만을 가져와 데이터를 주입, 가공하는 식의 개발 방법을 제공하여 UI 에서의 디자인코드(HTML)와 비즈니스로직(Javascript)을 분리. Controller에서 HTML오브젝트를 제어하기 위해, 디자이너는 HTML의 각 컴포넌트에 고유 식별아이디를 디자인 과정에서 부여	개발 생산성 유지 보수성

3.2 CVC 아키텍처 패턴의 정의

본 장에서는 기존 Ajax기반의 웹 응용 개발의 문제점을 개선하기위해 MVC 패턴 기반의 Ajax 웹 응용 개발을 위한 아키텍처 패턴을 정의 한다. 패턴의 명칭은 Communicator-View-Controller 의 약자인 CVC 패턴으로 명명하였고, 표 1의 개선 방안이 효과적으로 수행될 수 있도록 구조화 하였다.

3.2.1 CVC 아키텍처 모델

CVC 아키텍처 모델은, 아키텍처의 관점을 서버 기반 응용의 관점에서 클라이언트 브라우저 중심으로 전환하여 기존모델(J2EE Model 2)에서 단순 View영역으로 처리 되던 클라이언트(브라우저) 영역을 Model과 View 영역을 재귀적으로 포함하는 구조로 재-정의함으로써, Ajax 기반 클라이언트를 웹 어플리케이션 서버와 서버기술의 종속성으로부터 탈피시킨다. 다음 그림은 본 논문에서 제안하는 CVC 기반 Ajax 웹 응용 아키텍처 모델을 설명한다.

CVC 기반 Ajax 웹 응용 아키텍처 모델은, J2EE 패턴의 Model 2 아키텍처 모델이 MVC를 적용해 명시적으로 Model-View-Controller 영역을 구분했던 것

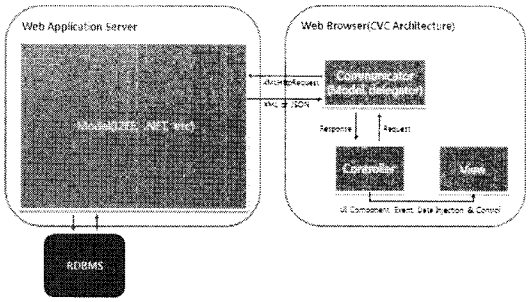


그림 2. CVC 기반 Ajax 웹 응용 아키텍처 모델
Fig. 2. Ajax architecture based on CVC model

과 유사하게, 웹 어플리케이션 서버와의 통신을 담당하는 Communicator 영역을 명시적으로 구분하고, 사용자 인터페이스를 담당하는 디자인 영역을 클라이언트 영역의 View로 재-정의함으로써 사용자 인터페이스를 위한 데이터와 이러한 데이터에 기반 한 비즈니스 로직을 사용자 인터페이스에 직접 연관된 디자인 영역으로부터 분리시킨다. CVC의 Communicatior는 그림 2의 Model영역 (서버 측)과 웹 브라우저 간 통신 업무를 담당하는 대리자 역할을 수행하고, 이를 통해 웹 브라우저단의 Controller에서는 Communicator를 통해 직접 서버 (서버 측 Model영역)를 다루게 되는 효과를 가진다.

3.2.2 CVC 아키텍처 패턴의 정의

본 장에서는 앞에서 정의한 CVC 아키텍처 모델을 CVC 아키텍처 패턴으로 정의 하였다. 다음은 CVC 아키텍처 패턴의 상세 명세(specification) 이다

(1) Communicator(Model delegator)

Communicator는 Controller로부터 전달받은 데이터를 웹 어플리케이션 서버의 Business Logic에 맞게 가공, 처리하여 데이터를 전달한다. 또한 전달받은 데이터 요청 쿼리에 대해 적절한 웹 어플리케이션 서버의 서비스를 선택하여 데이터를 추출하고, 이를 컨트롤러에 전달 해 주는 역할을 수행 한다. Controller와 웹 어플리케이션 서버 (서버 측 Model)와의 대리자 역할을 하며 J2EE 패턴 중 퍼사드(facade)[15] 패턴을 활용하여 서버가 수행하는 일을 Controller로부터 은닉하였다. 이를 통해 서버기술에 종속적이지 않은 HTML기반의 CVC 패턴을 구현 할 수 있다.

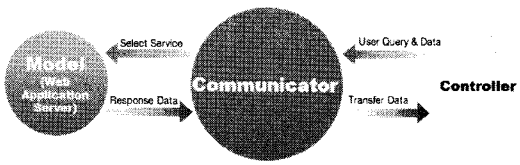


그림 3. CVC 패턴의 Communicator
Fig. 3. Communicator in CVC architecture pattern

(2) View

사용자가 보는 화면과 어플리케이션과의 인터페이스를 담당한다. 사용자의 요청이 있으면 Controller에 정의된 이벤트를 통해 사용자의 상태나 데이터를 서버에 전달한다. View 영역에서는 디자인 영역과 개발 영역의 완벽한 구분을 위해 어떠한 비즈니스 로직과 인라인스크립트[1]도 포함되지 않아야 한다. HTML컴포넌트의 이벤트와 데이터 주입 및 제어는 Controller 영역에서 모두 관장한다. 그림 4에 도시된 바와 같이 View 영역은 비즈니스로직으로부터 완전히 분리 되어 있으므로, 다른 영역의 개발과 독립적 개발이 가능하게 되고 이를 통해 개발 생산성 및 유지 보수성을 크게 증대시킬 수 있다. 이러한 View영역의 완전한 독립을 위해 본 논문에서는 MVC 패턴에서 Passive Model[2]방식을 적용하여 Model영역에서 View영역을 직접 제어 하지 못하도록 한다.

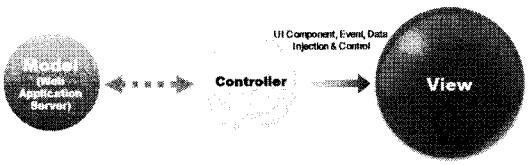


그림 4. CVC 패턴의 View
Fig. 4. View in CVC architecture pattern

(3) Controller

View의 객체 (텍스트, 콤보박스, 테이블 등)에 이벤트 등을 주입, 컨트롤 하고 View의 상태를 처리하며 사용자가 요청한 데이터를 Model로 전달하는 역할

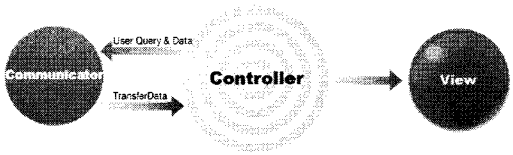


그림 5. CVC 패턴의 Controller
Fig. 5. Controller in CVC architecture pattern

1) HTML이나 XML 태그안에 Script가 포함되는 구문(HTML 태그안에 Javascript 가 포함되는 구문)
2) Passive model : Controller 가 이벤트를 받아서 처리하면서 Model 의 데이터를 변경함, Active model : Controller 의 관여 없이 Model의 데이터 변경

과 그 반대로 Model에서 데이터를 가져와 View의 객체에 주입해주는 역할을 수행한다.

Ⅳ. 실험 결과

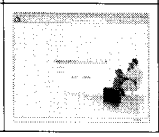
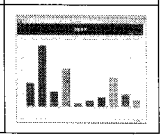
4.1 테스트 어플리케이션

CVC패턴의 효용성을 증명하기위한 테스트 어플리케이션으로, 본 논문에서는, 현행 Ajax 아키텍처 패턴 기반과 CVC 아키텍처 패턴 기반의 「회원관리 시스템」을 각각 개발 하였다. 각 시스템은, 공정한 비교를 위해, 데이터접근을 위한 서버 모듈 외에 JSP나 ASP등의 서버 페이지 기술을 배재하고 순수 클라이언트 기술 (HTML, Javascript, Ajax등)만을 활용하여 개발하였다.

4.1.1 화면구성

회원관리 시스템의 화면은 로그인과 회원가입을 할 수 있는 로그인 페이지, 회원 정보를 관리하는 회원관리 페이지, 로그인 현황을 볼 수 있는 회원통계 페이지로 구성된다. 하나의 View는 다이얼로그나 차트 등으로 표현되는데, 모든 다이얼로그는 DIV태그에 의한 레이어로 표현하였는데, 드래그 & 드롭이 가능하고 표시될 때와 사라질 때 에니메이션 효과가 나타나도록 하였다. 다음은 각 기능별 View의 구성이다.

표 2. 기능별 화면 구성
Table 2. Graphic User Interface of each Function

화면명	로그인	회원관리	회원통계
파일명	Login.html	MemberMgt.html	MemberStatistics.html
View	- 로그인, 로그인후, 회원가입 View	- 회원목록, 조회, 수정, 등록 View	- 일일, 월간, 년간 회원 로그인 통계 View - 일일, 월간, 년간 회원 로그인 순위 View
화면			

4.1.2 컴포넌트구성

개발의 표준화와 생산성을 위해 최신 Javascript Library 중하나인 jQuery[3]를 사용하여 Ajax를 구현하고 DOM을 컨트롤 하였으며, 회원통계의 차트를 표현하기 위해 jQuery기반의 Bar 차트 라이브러리[4]를 사용하였다. 또한, 유틸리티성격의 함수와 차트를 표 설정 값은 자바스크립트 파일로 따로 작성하여 사용하였는데, 표 3에 작성된 파일과 사용된 라이브러리 및 구현된 기능에 대산 설명을 나타내었다.

표 3. 컴포넌트 구성 및 기능 설명
Table 3. Constituent components and their function

라이브러리	파일명	설명
jQuery	jquery-1.5.min.js	자바스크립트와 Ajax를 간단하고 편리하게 구현 가능하게 도와주는 라이브러리
jQuery UI Lib	jquery-ui-1.8.9.custom.min.js	동적인 UI를 쉽게 구현할수 있게 해주는 라이브러리
jQuery Chart	wz_jsgraphics.js, graph.js	자바스크립트 기반의 차트를 쉽게 구현할 수 있도록 도와주는 라이브러리
Application Utility	common.js	어플리케이션 공용 유틸리티 라이브러리
Application Config	config.js	어플리케이션 전역 설정값 저장

4.1.3 서버 측 어플리케이션 구성

서버 측 어플리케이션은 Spring MVC 기반의 Java 어플리케이션으로 구현 하였으며, 조회되는 데이터는 JSON 형식의 데이터를 HTTP (Hyper Text Transfer Protocol)를 통해 전송하도록 하였다. 현행 Ajax 아키텍처를 사용하여 구현된 회원관리 시스템과 CVC 패턴을 기반으로 한 회원관리 시스템 모두 하나의 서버 측 어플리케이션으로 연계되도록 구성하였다.

4.2 평가결과

4.2.1 LOC(Line Of Code) 및 소스파일크기 비교

회원관리 시스템의 소스코드 중 공백 줄과 주석을 제외한 전체라인수와 소스파일의 크기를 측정, 비교한 결과를 표 4에 정리하였다. 표 4의 결과를 보면, 소스코드의 라인수가 아주 적게 보이는데 이는 jQuery를 사용했기 때문이다. jQuery는 복잡한 Ajax 호출문과 DOM Element 제어문, 화려한 UI를 간단하게 표현할 수 있는 기능 등을 제공하여 Javascript의 양을 대폭 줄여 주는 역할을 한다. 실제 회원관리 시스템을 jQuery없이 개발했다면 소스코드의 양은 10배가량 증가했을 것으로 예상된다.

3) The jQuery Project, http://jquery.com

4) jQuery Chart, http://www.sbmkpm.com/graph.html

표 4. LOC 및 소스파일크기 비교
Table 4. Comparisons of LOC and file size

<table>
<tr><th>현행 Ajax
(LOC, Byte)</th><th colspan="2">CVC(LOC, Byte)</th></tr>
<tr><td colspan="3">common.js(85, 2,077)</td></tr>
<tr><td colspan="3">config.js(20, 607)</td></tr>
<tr><td colspan="3">style.css(55, 1,358)</td></tr>
<tr><td>Login.html
(247, 8,130)</td><td>Login.html(85, 3,418)
LoginServiceController.js
(91, 2,720)</td><td rowspan="5">MemberCommunicator.js(197, 5,528)</td></tr>
<tr><td rowspan="2">MemberMgt.html
(441, 14,471)</td><td>MemberMgt.html
(152, 5,204)</td></tr>
<tr><td>MemberServiceController.
js(168, 5740)</td></tr>
<tr><td rowspan="2">MemberStatistics.
html(361, 11,050)</td><td>MemberStatistics.html
(115, 3,530)</td></tr>
<tr><td>MemberStatisticsServiceC
ontroller.js(161, 4,855)</td></tr>
<tr><td>1,209 line, 37,693 byte</td><td colspan="2">932 line, 29,509 byte</td></tr>
<tr><td colspan="3">LOC:22.91% 감소, 소스코드의 파일크기:1.71% 감소</td></tr>
</table>

평가의 공정성을 위해, 현행 Ajax 아키텍처에서 CVC 아키텍처로 변경 시 중복되는 기능에 대한 추가, 수정, 삭제 작업 없이 본 논문에서 제안하는 방식으로 구조만을 변경하였음에도, LOC는 22.91%가 줄었고, 소스코드의 파일크기는 21.71%가 줄었다. 이는 제안 아키텍처 패턴의 적용을 통해 개발 생산성 및 유지보수성이 크게 높아질 수 있음을 의미한다.

4.2.2 모듈의 재사용도

회원관리 시스템의 각 화면의 기능별 모듈의 재사용 횟수를 측정한 결과이다. 하나의 모듈이 2회 이상 사용될 때마다 1회씩 카운트를 더하였다. 공통모듈(common.js, config.js)의 재사용횟수는 양쪽 어플리케이션에서 공통으로 사용되는 모듈이므로 카운트에서 제외 하였다. 다음의 표 5에 이와 같은 모듈 재사용도에 대한 결과를 요약하고 추출된 재사용 모듈의 역할과 정의는 표 6에 정리 하였다.

표 5의 결과를 보면, CVC 패턴을 적용하여 시스템을 개발함으로써 모듈의 재사용도가 61% 이상 개선됨을 알 수 있는데, 이는 현행 Ajax기반 클라이언트 개발에서는 서버통신 모듈과, 비즈니스로직, UI컨트롤 로직 등이 모두 섞여있어 어플리케이션 모듈의 재사용도가 떨어지는 반면, 본 논문에서 제안하는 CVC 패턴을 사용하는 경우에는 각 기능별로 로직이 나뉘어져 있어 모듈의 재사용이 매우 용이하기 때문이다.

표 5. 어플리케이션 모듈별 재사용 횟수
Table 5. Count of reusage for each application module

<table>
<tr><th rowspan="2">화면명</th><th colspan="2">현행 Ajax</th><th colspan="2">CVC</th></tr>
<tr><th>모듈명</th><th>재사용횟수</th><th>모듈명</th><th>재사용횟수</th></tr>
<tr><td rowspan="8">어플리케이션모듈</td><td rowspan="8">doSelect
doList004</td><td rowspan="8">5
2</td><td>getSessionUserInfo</td><td>2</td></tr>
<tr><td>deleteSessionUserInfo</td><td>2</td></tr>
<tr><td>selectUserInfo</td><td>4</td></tr>
<tr><td>insertUserInfo</td><td>1</td></tr>
<tr><td>selectUserLoginLogSttstsList</td><td>2</td></tr>
<tr><td>doSelect</td><td>5</td></tr>
<tr><td>doList004</td><td>2</td></tr>
<tr><td></td><td></td></tr>
<tr><td>합계</td><td colspan="2">7 회</td><td colspan="2">18 회</td></tr>
<tr><td colspan="5">소스코드의 재사용도 61.11% 증가</td></tr>
</table>

표 6. 재사용된 모듈의 역할과 정의
Table 6. Functional definition of each reused module

모듈명	모듈구분	입력	수행작업내용	출력
getSessionUserInfo	Communicator	없음	로그인된 사용자 정보를 서버로부터 읽어온다.	사용자 정보
deleteSessionUserInfo	Communicator	없음	로그인된 사용자 정보를 제거(로그아웃)한다.	없음
selectUserInfo	Communicator	사용자 아이디	사용자 정보를 서버로 부터 읽어온다.	사용자 정보
insertUserInfo	Communicator	사용자 정보	회원가입을 처리한다.	없음
selectUserLoginLogSttstsList	Communicator	기준년,월,일	일일, 월간, 년간 회원로그인 통계 데이터를 서버로부터 가져온다.	일일, 월간, 년간 회원로그인 통계 데이터
doSelect	Controller	회원 아이디	Dialog박스(jQuery UI)에 회원정보를 표시해 준다.	회원정보 Dialog
doList004	Controller	기준년,월,일	회원로그인 순위 차트를 만들어 표시 해 준다.	Bar 차트

4.2.3 소스코드의 중복도

다음은 회원관리 시스템 소스코드의 중복성을 측정한 결과이다. 소스코드의 중복성은 기 개발된 코드의 유지 보수 시 그 편의성을 제공하는데 매우 중요한 요소 중 하나이며, 본 논문에서는 전체 어플리케이션 소스 중 1개의 함수 안에 소스코드가 5라인 이상 중복된 건수만을 측정하였다. 소스코드가 중복되는 경우는 보통 여러페이지에 같은 함수가 똑같이 존재하는 경우

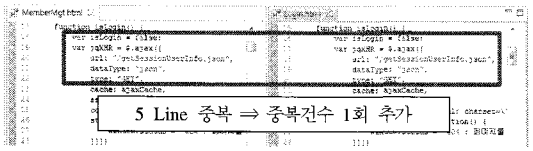


그림 6. 현행 Ajax 모델의 MemberMgt.html 과 Login.htm 파일의 중복된 소스 예
Fig. 6. An example of source redundancy in Ajax model

나 서버와 통신하기 위한 Ajax 관련 구현체 들이다. 다음은 실제 중복된 건수 1건에 대한 예시이다.

다음은 위와 같은 방법으로 산출한 소스코드의 중복도 측정 결과이다.

표 7에서 볼 수 있듯이, 본 논문에서 제안하는 CVC 패턴을 적용함으로써 소스코드의 중복도가 19회에서 13회로 약 32% 개선되었다. 13 회의 중복건수 중 대부분은 모듈별로 분리되면서 자연스럽게 중복을 제거 할 수도 있었지만 평가의 공정성을 위하여 수정하지 않고 중복횟수에 추가 하였다.

표 7. 소스코드의 중복도
Table 7. Redundancy of source code

현행 Ajax	CVC
19 회	13 회
소스코드의 중복도 31.57% 감소	

4.2.4 디자인코드와 개발코드의 구분도

제안된 CVC 패턴의 View영역의 구현을 위한 디자인 코드와 개발자 코드의 구분도를 측정하기 위해, LOC결과의 산출에서와 마찬가지로 공백 줄과 주석을 제외한 후, <head> 태그 안에 스타일시트를 로드하기 위한 <link>태그를 디자인코드로 정의하고, 자바스크립트를 로드하고 개발하기 위한 <script> 태그를 개발코드로 정의하였다. 또한, HTML 코드 안에 포함되어 있는 인라인 스크립트는 개발코드 1 라인으로 가정하였으며, 디자인 코드는 <body>태그 내의 라인수를 카운트 하였다. 표 8에 측정 결과를 나타내었다.

표 8의 결과에서 볼 수 있듯이, 본 논문에서 제안하는 CVC 패턴을 적용함으로써 디자인 페이지인 View 페이지 안에 개발 코드가 7.8%만 존재하게 되었는데, 이마저도 ServiceController.js, Communicator.js와 기타 필수 라이브러리를 로드하기 위한 구문일 뿐 실제 비즈니스로직은 아니다. 따라서 본 논문에서 제안하는 아키텍처 패턴을 적용함으로써 디자인 페이지 내 개발 코드가 실제로 100% 제거 되었다고 볼 수 있으며, 이로써 UI컴포넌트의 위치나 색상 등이 바뀌어도 해당 컴포넌트가 존재하고 속성만 바뀌지 않는다면 개발코드와는 별개로 처리될 수 있어 매우 높은 개발 생산성과 유지보수성 향상을 얻을 수 있다.

표 8. 디자인페이지의 개발코드 비율
Table 8. Ratio of development codes in a design page

View 파일명	현행 Ajax(Line수)	CVC(Line수)
Login.html	디자인 : 66 / 개발 : 168	디자인 : 66 / 개발 : 7
MemberMgt.html	디자인 : 133 / 개발 : 292	디자인 : 133 / 개발 : 7
MemberStatistics .html	디자인 : 94 / 개발 : 252	디자인 : 94 / 개발 : 9
합계	디자인 : 293 / 개발 : 436	디자인 : 293 / 개발 : 23
비율	디자인 : 32.8% / 개발 : 67.2%	디자인 : 92.2% / 개발 : 7.8%
	디자인페이지의 개발코드 67.2 % 에서 7.8%로 감소	

V. 결 론

본 논문에서는 복잡한 Ajax 기반 웹 응용 개발에서 나타나는 개발 생산성 및 유지 보수성의 저하라는 문제점을 해결하기 위해, MVC 패턴 기반의 CVC 아키텍처 패턴을 제안하였다. CVC 아키텍처는 Ajax 기반 클라이언트가 공통적으로 가지게 되는, 데이터 추출을 위한 비동기 통신 관련 코드를 Communicator 영역으로 분류함으로써, 그래픽 디자인 영역에 해당하는 View와, View 계층에 효과적으로 데이터를 갱신하는 Controller 영역으로 Ajax 기반 클라이언트 코드를 분류한다.

제안 아키텍처의 효용성을 증명하기 위해, 본 논문에서는 Ajax 기반 회원관리 시스템을 구현하였는데, 구현된 시스템은 로그인 (인증), 회원관리 (게시판형식의 CRUD[5]), 회원통계 (레포트)등 웹페이지의 대표적인 필수 요소들을 모두 포함하고 있다. 이러한 회원관리 시스템의 구현에 제안한 CVC 아키텍처 패턴을 적용한 결과 디자인코드 내 개발코드의 비율 및 소스코드의 중복도와 모듈의 재사용도, 소스파일의 크기 등에 있어 매우 큰 개선효과를 얻을 수 있었는데, 우선 디자인코드 내 개발코드의 비율이 기존 디자인페이지 (HTML)의 경우 67.2%를 차지하던 것이 7.8%로 감소하는 것을 확인할 수 있었다. 여기서, 이 7.8%의 개발소스 부분도 실제로는 HTML이 파일을 직접

5) CRUD(Create, Read, Update, Delete) : 입력, 조회, 수정, 삭제의 웹 환경에서 일반적으로 수행하는 사용자의 동작을 가리킨다.

제어 할 수 없는 기능상의 한계로 인해 개발로직을 Import하는 구문일 뿐 비즈니스로직과 전혀 관계없는 구문에 해당하므로, 디자인페이지 내에 개발소스 (비즈니스로직)가 전혀 포함되지 않았다고 볼 수 있으며, 이러한 결과는 디자인 영역에 대한 독립성을 보장해 줌으로써 디자이너의 개발생산성을 크게 증대 시킬 수 있고, 따라서 디자인 변경사항에 대한 유연한 개발 및 유지보수 환경을 제공해 주며, 개발자와 디자이너 간의 업무책임에 대한 분쟁을 최소화 해주는 효과를 가져다준다. 또한, 소스코드의 중복도와 모듈의 재사용도 측정을 통한 결과에서는 소스코드의 중복도가 낮아지고 모듈-간 독립성이 보장되어 모듈의 재사용도가 61% 이상 개선되었음을 확인 할 수 있었는데, 이러한 결과는 소스코드의 중복도가 최소화됨으로써 소스코드가 단순해지고 가독성이 좋아져 유지보수성이 증가되고, 모듈을 재-사용함으로써 개발시간을 단축시켜 개발생산성 향상에 큰 도움을 주게 된다. 마지막으로, LOC 와 소스파일의 크기의 비교 측정에서는, 소스코드자체가 jQuery에 의해 많이 최적화 되어 있어 최적화된 소스에서는 차이가 작게 나온다는 LOC 측정의 특성을 감안하더라도, 22% 내외의 높은 개선효과가 나타났는데, 이러한 LOC의 개선효과는 개발하는 응용의 크기가 커질수록 더욱 커질 수 있어 제안 아키텍처를 통한 개발 생산성의 매우 높은 향상을 의미한다 할 수 있다.

본 논문에서 제안한 CVC 아키텍처 패턴은 고급 사용자 경험을 제공하는 복잡한 Ajax 기반 클라이언트 개발의 방법을 서술적으로 개념화시킴으로써, Ajax 관련 코드를 효과적으로 모듈화 하여 재사용하고 그래픽 디자인을 독립적으로 처리할 수 있도록 하여 Ajax 기반 웹 응용의 개발생산성 및 유지보수성을 획기적으로 높일 수 있도록 하였다. 이러한 CVC 아키텍처는 순수 표준 기술만으로 구현이 가능한 패턴이므로, 기존 HTML의 한계로 인한 서버기술에 대한 개발 종속성을 벗어날 수 있는 기술이다. 하지만 이를 보다 전문적으로 활용하기 위해서는 제안 CVC 패턴에 최적화된 아키텍처 프레임워크가 구현되어야 할 필요가 있으며, 웹 UI의 여러 요소들이 모두 포함된 아키텍처 프레임워크의 제공이 본 논문의 기술을 상용화하기 위한 향후 연구 방향이라 할 수 있다.

참 고 문 헌

[1] 이성혜, "UI는 보여지는 것만을 위한 작업이다?," 마이크로소프트웨어, p.33, 7월 2003년.

[2] Wikipedia, "Ajax (programming)," May 2011 [Online]. Available: http://en.wikipedia.org/wiki/Ajax

[3] James Y. Xu, "Integrating REST and Ajax into Model-View Controller - A web based billing system case study using Python," IEEE IT Professional, issue 99, p.1, 2010.

[4] K. Samkari and A. Joukhadar, "Comparison matrix for web HCI", in Proc. of Int. Conf. on ICT, 2008, pp.1-5.

[5] X. Liu, L. Liao, Y. Duan, and B. Yang, "Email information integration with SSO in portal serviced based on Ajax," in Proc. of Int. Conf. on CASM, 2010, pp.544-548.

[6] H. Yang, J. Shi, and X. Zhang, "The update version development of 'Wiki Message Linking' system - Integrated Ajax with MVC model," in Proc. of Int. Forum on CSTA, Dec. 2009, pp.209-212.

[7] H. Song, M. Zhang, and Z. Xu, "Design and implementation of online stock trading system," in Proc. of Int. Conf. on CISE, Dec. 2009, pp.1-4.

[8] H. Wang, Q. Zhu, J. Shen, and S. Cao, "Web-service-based design for rural industry by the local e-government," in Proc. of Int. Conf. MINS, Nov. 2010, pp.230-235.

[9] Z. Wang, Z. Liu, and Y. Yang, "Design and implementation of flexible e-government platform based on XML data-bus and lightweight MVC execution framework ASSH," in Proc. of Int. Conf. on CASM, Oct. 2010, pp.422-427.

[10] Wikipedia, "Model-view-controller," May 2011 [Online]. Available: http://en.wikipedia.org/wiki/

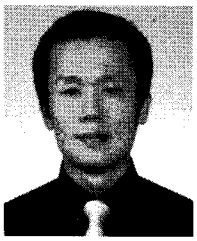
[11] A. Goldberg, "Smalltalk80: The interactive programming environment", Addison-Wesley Publ., 19894.

[12] Li Yan, "Intelligent multi-agent user interface design," in Proc. of IEEE Int. Forum on ITA, 2009, pp.496-498.

[13] J. Stocklein, C. Geiger, V. Paelke, and P. Pogscheba, "Poster: MVCE - A design pattern to guide the development of next generation user interfaces," in Proc. of IEEE Sympo. on

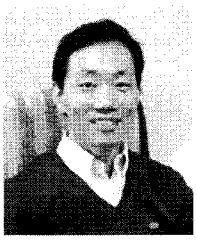
3D User Interfaces, 2009, pp.153-154.
〔14〕 Seshadri G, Understanding JavaServer Pages Model 2 architecture, JavaWorld.com [On-line], Available: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
〔15〕 Gamma E., Helm R., Johnson R., and Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Boston: Addison-Wesley, 1994.

김 황 만 (Hwang-man Kim) 준회원



2004년 2월 호원대학교 컴퓨터공학 학사
2011년 8월 KGIT 미디어공학 석사
<관심분야> 웹, S/W 아키텍처, S/W 디자인 패턴

김 용 구 (Yong-Goo Kim) 종신회원



1993년 2월 연세대학교 전기공학과 학사
1995년 8월 연세대학교 전기및 컴퓨터공학과 석사
2001년 8월 연세대학교 전기전자공학과 박사
2002년 9월~2006년 2월 (주) 온타임텍 연구소장
2009년 3월~현재 KGIT 뉴미디어학부 교수
<관심분야> 비디오 압축/ 통신, 멀티미디어 신호 처리 및 응용