ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

성능 주도의 UI-Mashup 아키텍처의 설계 및 구현

Design and Implementation of the Performance Driven UI-Mashup Architecture

조 동 일[1*]
Dong-Il Cho

요 약

UI-Mashup은 웹 응용프로그램 개발의 최신 경향 중 하나로 인터넷 상에 분산된 다양한 콘텐츠를 조합하여 가치를 추가해 서비스하는 방안으로 널리 사용되고 있다. 현재까지 UI-Mashup 관련 연구는 동적 서비스 조합에 초점이 맞추어져 있고 급변하는 웹 표준에 적응하지 못하여 최종 사용자 입장에서 UI-Mashup은 느리고 불편하며 보안에 취약한 서비스로 인식되고 있다. 본 연구에서는 UI-Mashup의 성능 향상을 위한 아키텍처를 제안한다. 제안한 아키텍처는 빠른 서비스 제공과 보안 강화를 위해 UI조각을 서버에서 병렬로 수집하고 매쉬업된 UI의 레이아웃과 UI조각들을 별도의 전송 채널을 통해 클라이언트로 전송하여 빠른 반응시간과 응답시간을 제공한다. 본 연구에서는 제안한 아키텍처를 실증적으로 검증하기 위해 구현하였으며 성능테스트를 진행하였다. 성능테스트 결과 제안한 아키텍처는 기존 UI-Mashup 기법에 비해 2 ~ 3배 빠른 응답시간을 기록하였고, 4배이상의 처리량을 보였다.

☞ 주제어 : UI-매쉬업, 기업포털, 서비스조합

ABSTRACT

UI-Mashup is widely used as a service method to add value, which is composed of distributed various contents on the internet and has turned out to be one of the latest trends in web application program development. Previous UI-Mashup-related studies have focused primarily on the dynamic service composition and have not been able to adapt to a rapidly changing Web Standard, thusthe end users conclude that UI-Mashups are slow, incompatible and poor security services. In this study, We propose an architecture for the performance improvements of UI-Mashup.In order to provide fast services and security enhancements, the proposed architecture collects UI fragments on the server in parallel, and sends layouts and contents of Mashups UI to the client through a special delivery channel supporting fast reaction and response time. In this study, the implementation and performance tests were proceeded to verify the proposed architecture experimentally. As a result of the performance testing, the proposed architecture has two to three times faster response time and more than four times throughput compared to the previous UI-Mashup technology.

☞ keyword : UI-Mashup, Enterprise Portal, Service Composition

1. 서 론

UI-Mashup은 웹 응용프로그램 개발의 최신 경향 중 하나이며 인터넷 상에 분산된 다양한 콘텐츠를 조합하여 가치를 추가해 서비스하는 방안으로 널리 사용되고 있다[1,5].

UI-Mashup은 큰 이점이 있지만 네트워크를 통해 분산된 다양한 자원을 수집하는 과정에서 서버 및 클라이언트의 많은 자원을 소모하며 느린 응답속도를 보인다[6]. 현재까지 UI-Mashup에 관한 연구는 동적 자동화된 서비스 조합에 관한 연구가 주를 이루고 있고 성능에 대한 연구는 단순한 사례 연구에 머물러 있다[14,18]. 때문에 UI-Mashup은 최종사용자 환경에서는 여전히 느리고 불편하다.

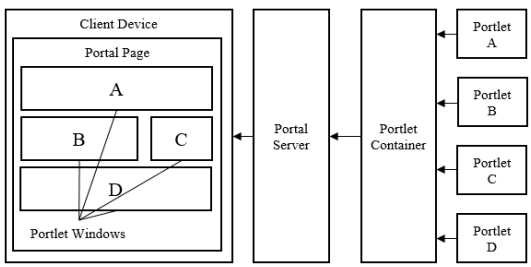
본 연구에서는 웹 기반 UI-Mashup에서 최종 사용자가 서버로 매쉬업된 UI를 요청했을 때부터 UI가 사용자에게 보여지기까지의 시간을 단축할 수 있는 아키텍처를 제안한다. 이를 위해 본 연구에서는 먼저 현재 연구되고 있는 UI-Mashup 방법들의 성능 문제에 대해 분석하고 성능 향상을 위한 매쉬업 방법 및 콘텐츠 전송 방법을 제시한다. 그리고 개선된 성능을 입증하기 위해 제시한 아키텍처를 설계하고 구현한 후 기존에 널리 사용되고 있는 기법과 성능을 비교하였다.

1 R&D Center, Tomatosystem, Seoul, Korea.
* Corresponding author (chodongil@yahoo.co.kr)
[Received 2 November 2013, Reviewed 12 November 2013, Accepted 27 December 2013]

2. UI-Mashup

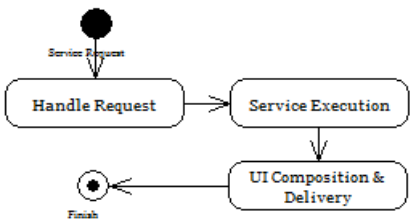
2.1 UI-Mashup 개요

UI-Mashup은 이미 존재하는 여러 서비스를 조합하여 사용자가 원하는 가치를 추가한 새로운 서비스를 생성한다[15]. 매쉬업된 UI는 여러 개의 UI조각으로 구성되고 하나의 UI조각은 각각 분산된 서비스 제공자로부터 수집된다[13]. 이 과정에서 UI조각을 생성하는 각각의 서비스는 상호 독립적으로 실행되며 다른 서비스의 영향을 받지 않는다. 대표적인 UI-Mashup 미들웨어인 포털 시스템은 정보와 응용프로그램의 단일화된 접점으로 모든 정보를 아울러 포괄적인 컨텍스트와 집계된 뷰를 제공한다[2]. 포털 페이지는 포틀릿이라고 불리는 UI조각들로 구성된다. 각 포틀릿은 독립적인 프로세스 실행을 통해 포틀릿 UI를 생성하고 포털 엔진은 이것들을 포틀릿 윈도우로 조직화하여 포털 페이지를 구성한다[3].



(그림 1) 포털시스템에서의 UI 매쉬업
(Figure 1) UI-Mashup at the Portal System

UI-Mashup은 다음 그림과 같은 절차로 수행된다[14,19,20].



(그림 2) UI-Mashup 절차
(Figure 2) UI-Mashup Workflow

클라이언트에 의해 Mashup-UI가 요청되면 매쉬업 서버에 의해 요청이 처리된다. 매쉬업 서버는 요청처리 과정에서 UI를 구성할 UI조각 목록과 권한 등을 식별한다.

다음으로 화면을 구성할 서비스를 실행하여 UI조각을 수집한다. 이 과정은 대부분 SOAP 등과 같은 원격 서비스 호출을 통해 수행된다.

수집된 각 UI조각은 필요에 의해 추가적인 가공을 거친 후 전체 UI의 한 부분으로서 조합되어 클라이언트에게 전송된다.

2.2 성능 향상을 위한 문제점

사용자가 느끼는 시스템의 속도는 시스템의 반응시간과 응답시간으로 구분할 수 있다. 반응시간은 최초 사용자가 이벤트를 발생시킨 이후 시스템의 최초 응답이 전달되기까지의 시간이고 응답시간은 마지막 패킷이 사용자에게 전달되기까지의 시간으로 정의된다[9].

UI-Mashup에 관한 연구는 UI조각의 수집과 매쉬업이 이루어지는 장소에 따라 서버 측 매쉬업과 클라이언트 측 매쉬업 그리고 이 둘을 필요에 의해 조합한 하이브리드 매쉬업으로 나눌 수 있다.

클라이언트 측 매쉬업은 UI조각을 위한 서비스 호출과 콘텐츠의 조합을 클라이언트에서 처리한다[1,4,7,8,10,18]. 주로 JavaScript와 같은 클라이언트 측의 프로그래밍 언어나 브라우저 플러그인을 이용하여 콘텐츠 제공 서버에 서비스를 호출하고 수집된 콘텐츠를 클라이언트 측에서 조합한다. 이 방법은 UI-Mashup 구현을 쉽게 하고 사용자가 느끼는 시스템의 반응시간을 단축시킬 수 있다. 또한 클라이언트 측의 병렬적인 콘텐츠 수집을 통해 전체적인 응답시간을 단축시킬 수 있다. 하지만 브라우저에서 동시에 서버로 요청할 수 있는 요청수의 제한과 새로운 네트워크 소켓을 연결하기 위한 시스템 부하 그리고 Same-Origin Policy(SOP)에 기인한 Cross-Domain 접근 제한 등의 문제로 인해 기능이 제한되고 유연하지 못하며 전체적인 시스템의 응답시간을 저하시킨다. 또한 대부분의 기능이 클라이언트에서 동작하기 때문에 잠재적인 보안문제를 가지고 있다[1,16].

서버 측 매쉬업은 콘텐츠의 수집과 매쉬업을 서버에서 처리한다. 서버에서 매쉬업을 처리하기 때문에 클라이언트 측 매쉬업에 비해 보안이 우수하고 강력한 기능을 제공할 수 있으며 UI조각 조합에 유연하다. 또한 SOP로부터 자유로워 보다 강력한 보안 정책의 적용이 가능하다. 하지만 콘텐츠의 수집이 서버에서만 이루어지기 때문에 서버에 많은 부하를 주어 동시 사용자 수에 제한이 있고, 응답속도가 느리고 사용자와의 상호 작용성이 떨어지는

단점이 있다[1].

하이브리드 매쉬업은 매쉬업을 서버와 클라이언트가 나누어서 처리하는 방법으로 여러 가지 아키텍처가 가능하다. Jian Meng의 연구는 UI레이아웃과 UI조각을 분리하여 사용자에게 서비스한다는 측면에서 본 연구와 유사성이 있다. 하지만 순차적인 콘텐츠의 수집과 XML로의 데이터 전환 그리고 XML를 다시 XSLT를 통한 UI조각으로의 변환 등으로 인해 처리 성능을 보장할 수 없으나 이 부분에 대한 검증을 하지 않았다[1]. 반면 Xuanzhe Liu과 Nassim Lage의 연구는 SOAP 프로토콜을 이용한 매쉬업을 제안한다[17]. 전형적인 클라이언트 측 매쉬업을 이용하여 브라우저에서 구동하는 Javascript 컴포넌트를 이용하여 웹 서비스를 통하거나 iFrame과 같은 기술을 이용하여 콘텐츠를 수집하여 브라우저에서 조합한다. 이 아키텍처는 SOP로부터 자유롭지 못하고 전체적인 QoS를 보장할 수 없다.

이와 같이 현재까지의 매쉬업 관련 연구는 자동화된 매쉬업 기법 제공에 치중하고 있다. 일부 연구에서는 성능 향상 기법을 제시하고 있긴 하지만SOP, 모바일 기기에서의 서비스 등의 문제로 인해 현실적이고 범용적인 사용자관점에서의 성능의 보장이 명확히 제시 및 증명 되지 못하고 있다.

3. 고성능을 위한 UI-Mashup 방안

3.1 UI조각 수집 방안

UI-Mashup을 위해 수집되는 UI조각들은 독립적인 서비스 호출을 통해 획득된다. 만약 이들 UI조각들을 수집하기 위한 서비스 실행이 순차적으로 이루어 진다면 전체 서비스 제공 시간(Z), 개별 UI조각 실행 시간(X), UI조각의 개수(n), UI 조직화 시간(Y) 이라고 했을 때 일반적인 전체 서비스 제공 시간은

$$Z(p) = \sum_{i=0}^{n} X(i) + Y(n)$$

라는 함수가 성립한다.

여기에서 개별 UI조각을 획득하기 위한 함수 X는 각 서비스들 간에 간섭이 없기 때문에 병렬 수행이 가능하다. 함수 X를 병렬로 수행할 경우 수식은

$$Z(p) = Max(X(0 \sim n)) + Y(n)$$

와 같이 변경할 수 있다. 즉 전체 UI-Mashup이 완료되는 시간은 하나의 UI조각이 만들어지는 최대 시간에 비례하는 성능 향상 효과를 기대할 수 있다.

3.2 UI-Mashup 및 전송 방안

클라이언트 측 매쉬업에서 서버의 페이지 레이아웃 서비스와 클라이언트의 UI조각 수집은 반응시간을 줄이는데 큰 효과가 있다. 하지만 클라이언트에서의 iFrame 또는 AJAX를 이용한 UI조각 수집은 전체적인 수행 성능을 저하시키고 SOP와 같은 보안적인 문제를 가지고 있다. 특히 모바일 환경과 같이 상대적으로 네트워크 소켓 연결 개수 제한이 심한 환경에서의 서비스까지 고려한다면 콘텐츠의 수집과 조합을 클라이언트에서 수행하는 것은 신뢰적인 서비스 제공을 어렵게 한다. 따라서 성능 향상과 보안 요소를 고려한다면 UI 조합은 클라이언트에서 이루어져야 하고 UI조각의 수집은 서버에서 이루어져야 한다.

그리고 UI조각의 전송 방법에 있어 화면을 구성하는 모든 UI조각을 수집하여 한번에 전송하는 것은 모든 UI조각의 수집이 완료될 때까지 대기해야 하기 때문에 반응시간을 느리게하는 문제점이 있다. UI조각의 수집 과정이 대부분 원격 서비스 호출을 통해 이루어지는 매쉬업 환경에서 네트워크 문제 등으로 인해 처리지연이 발생할 수 있는데 수집된 UI조각들을 한번에 클라이언트로 전송한다면 모든 UI조각의 수집이 끝날 때 까지 사용자는 대기해야 하기 때문에 QoS를 보장할 수 없다. 반면 UI조각 별로 독립적인 프로세싱을 통해 수집하고 먼저 수집된 순서로 클라이언트로 전송한다면 문제가 발생된 것을 제외한 나머지는 서비스를 제공할 수 있기 때문에 서비스의 품질이 높아지는 효과를 제공할 수 있다. 따라서 반응시간 단축을 위해서는 UI조각의 전달은 수집이 끝난 UI조각 순으로 클라이언트에 전달할 수 있는 별도의 전송 채널이 필요하다.

4. 고성능 UI-Mashup 아키텍처 설계

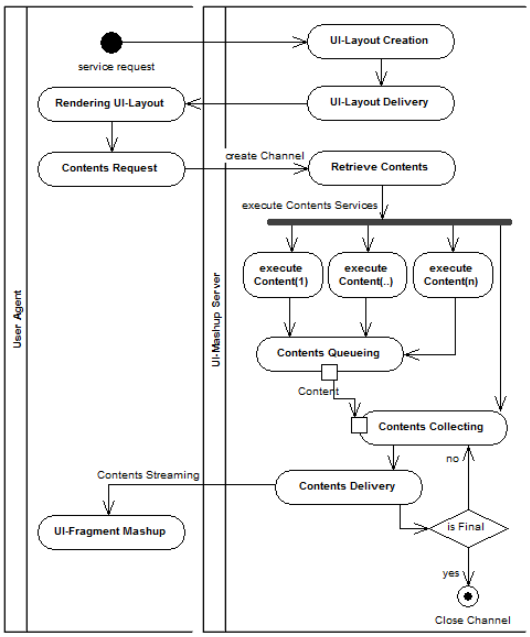
4.1 고성능을 위한 UI-Mashup 시나리오

본 연구에서 제안하는 고성능을 위한 UI-Mashup 아키텍처의 주요 아이디어는 다음과 같다.

- UI레이아웃과 UI조각의 분리 전송

- UI조각의 병렬수집
- 비동기적 UI조각 전송 채널

이들 아이디어를 이용한 UI-Mashup 아키텍처는 다음과 같은 절차로 수행된다.



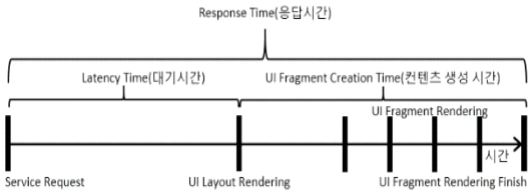
(그림 3) 고성능을 위한 UI-Mashup 액티비티 다이어그램
(Figure 3) UI-Mashup Activity Diagram for High Performance

1. 클라이언트는 서버로 서비스를 요청한다.
2. UI-Mashup 서버는 화면을 구성할 UI레이아웃을 생성하여 클라이언트로 전송한다.
3. 클라이언트는 전달 받은 UI레이아웃을 화면에 렌더링한다.
4. 클라이언트는 UI레이아웃 내부의 UI조각들을 서버로 요청한다. 이때 서버와 클라이언트 간에는 UI조각을 전송 받은 수 있는 전송채널이 형성된다.
5. 요청을 받은 서버는 클라이언트에 전달할 UI조각 목록을 생성한다.
6. 서버는 UI조각을 수집할 스레드와 수집된 UI조각을 클라이언트로 전송할 서비스를 생성하여 독립적으로 실행한다. UI조각 수집 스레드들은 서버에서 병렬로 실행되며 수집되는 UI조각들은 수집된 순서대로 대기열에 쌓인다.
7. UI조각을 전송할 스레드는 대기열에 UI조각이 추가될 때까지 대기 상태로 있다가 UI조각이 추가되면 클라이언트로 전송한다.
8. UI조각을 수신한 클라이언트는 전달 받은 UI조각을 화면에 추가하고 다음 UI조각을 수신을 위해 대기한다.
9. UI-Mashup 서버의 UI조각 전송서비스는 모든 UI조각의 수집과 전송이 완료되면 클라이언트와의 전달 채널을 종료하고 서비스를 종료한다.

4.2 UI레이아웃과 UI조각의 분리 전송

UI-Mashup 시스템이 사용자의 요청에 빠르게 반응하기 위해서 사용자와 상호작용을 위한 대기시간을 줄이는 것이 중요하다. 대기시간은 실행될 작업이 존재하지 않더라도 어떤 형식의 반응을 얻기 위해 요구되는 최소한의 시간이다[9].

UI-Mashup에서 UI조각을 수집하는 시간은 콘텐츠의 특성에 따라 네트워크를 통한 원격 호출로 인해 많은 대기시간을 필요로할 수 있다. 반면 UI레이아웃은 사용자가 요청한페이지에 대응하는 화면을 구성하는 뼈대로 서버의 메타정보만으로 구성이 가능하다. 따라서 UI레이아웃과 UI조각을 분리 전송하는 것으로 사용자 요청에 기민하게 반응하는 서비스를 제공해 줄 수 있다.



(그림 4) 대기시간과 응답시간
(Figure 4) Latency Time & Response Time

따라서 제안하는 아키텍처는 UI레이아웃과 UI조각을 별도의 전송 채널을 이용하여 클라이언트에 전송하고 클라이언트는 전달 받은 콘텐츠를 실시간으로 렌더링하여 사용자에게 시스템이 반응하고 있음을 시각적으로 표현해준다.

4.3 UI조각의 병렬 수집

매쉬업 화면을 구성할 UI조각을 생성하는 서비스들은 상호 간에 간섭을 받지 않고 독립적으로 수행된다. 이들

서비스를 순차적으로 실행하는 것은 시스템 전체의 수행 시간을 증가 시켜 시스템의 효율을 떨어뜨린다. 따라서 콘텐츠 서비스를 병렬로 수행함으로써 전체적인 시스템 효율성을 증대시킬 수 있다.

본 연구에서는 UI조각 서비스의 병렬 실행을 위해 서비스 병렬 실행 엔진을설계 및 구현하였다. 서비스 병렬 실행 엔진은서비스 실행 스레드를 풀로 관리하여 효율성을 높이고 제한된 시스템 자원만을 사용하도록 하여 신뢰성을 제공한다. 또한 사용이 완료된 스레드는 다시 풀로 환원되어 다음 처리에 재활용된다.

서비스 병렬 실행 엔진은 각각의 UI조각 수집 서비스를 독립적으로 실행하여 서비스 별 QoS를 관리할 수 있으며, 내부 Socket 자원을 재활용하여 외부 시스템 호출에 높은 성능을 보장한다.

4.4 비동기적 UI조각 전송 채널

본 연구에서 제안한 UI조각의 수집을 서버에서 그리고 UI조각의 조합을 클라이언트에서 처리하기 위해서는 서버에서 수집된 UI조각을 클라이언트로 전송하기 위한 방법이필요하다. HTTP 환경에서 하나의 요청은 하나의 응답에 대응하기 때문에 일반적인 HTTP를 이용한다면 서버에서 수집된 UI조각을 클라이언트로 전달하기 위해서 UI조각의 개수 만큼의 HTTP 연결이 필요하고 이것은 클라이언트와 서버에 많은 부담 주며 빈번한 서버 연결로인해 처리성능을 저하시킨다.

이 문제를 해결하기 위해 본 연구에서는 Reverse AJAX를 이용해 서버에서 클라이언트로 푸시방식의 UI조각 전송을 이용한다. Reverse AJAX는 서버에서 클라이언트의 Javascript 함수를 능동적으로 호출하는 방법이다[11].

제안한 아키텍처에서는 UI조각의 조합을 서버에서 UI 레이아웃과 함께 전송된 UI 조합 Javascript 모듈의 함수를 실행하는 스크립트를 Reverse AJAX를 통해 전송하여 동적으로 화면을 클라이언트에서 조합한다. 이 기술은 UI조각 수집으로 인한 대기시간을 최소화하고 서버와 클라이언트 간의 연결자원을 하나만 사용하여 모든 UI조각을 전송할 수 있어 높은 성능을 제공한다.

4.5 보안 및 접근 제어

UI-Mashup은 여러 개의 UI조각들을 수집하여 하나의 화면을 생성한다. 개개의 UI조각은 접근이 인가된 사용자에게만 서비스 되어야 하고 그렇지 않은 사용자에게는 접근이 제한되어야 한다.

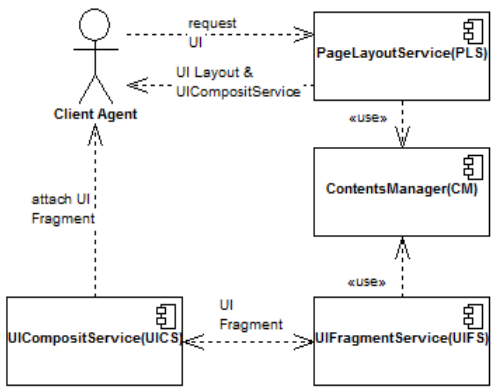
UI조각의 수집과 조합을 클라이언트에서 할 경우 이런 접근 제어는 어렵거나 불가능하다. AJAX를 통한 UI 수집은 SOP에 의해 수집할 수 있는 서버가 제한적이고 브라우저를 통해 코드가 노출되어 보안에 불리하다[12].

본 연구에서 제안한 UI-Mashup 아키텍처는 UI조각의 수집을 서버에서 처리하여 UI조각 각각의 권한 및 접근 통제가가능하다. 또한 하나의 채널을 통해 서버 푸시 방식으로 UI조각이 클라이언트로 전달되기 때문에 코드 노출로 인한 보안 문제가 발생하지 않는다.

5. 구현 및 성능 평가

5.1 구현

시스템은 현재 가장 널리 사용 중인 언어인 Java 언어로 구현하였다. 시스템은 크게 네 개의 컴포넌트로 구성되며 다음 그림은 UI-Mashup 아키텍처의 컴포넌트 다이어그램이다.



(그림 5) 컴포넌트 다이어그램
(Figure 5) Component Diagram

ClientAgent(CA)는 사용자가 서비스 요청에 사용하는 응용프로그램으로 브라우저가 이에 해당한다.

PageLayoutService(PLS) 컴포넌트는 사용자의 요청을 받아 처리하는 컴포넌트이다. 이 컴포넌트는 사용자가 요청하는 UI의 구성을 ContentsManager(CM)를 통해 조회하여 UI레이아웃을 생성한다. 생성된 UI레이아웃은 HTML과 JavaScript, 그리고 CSS 와 같은 전형적인 웹 콘텐츠로 구성되고 매쉬업 될 UI조각들이 위치할 부분이 정해져 있다. PLS 컴포넌트는 생성한 UI레이아웃을 UICompositService(UICS) 컴포넌트와 함께 CA로 전송한다.

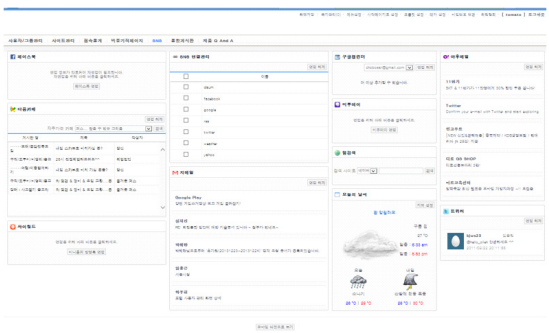
UICS 컴포넌트는 순수 JavaScript로 구성된 컴포넌트로 CA에서 실행된다. UICS 컴포넌트는 서버에서 구동하는 UIFragmentService(UIFS) 컴포넌트와 통신 채널을형성하고 UIFS 컴포넌트로부터 Reverse AJAX 형태로 UI조각의 조합 서비스가 호출된다. UICS 컴포넌트에는 UI조각의 화면 배치, UI조각에서 사용하는 JavaScript 또는 CSS 등의 자원 관리 그리고 UI조각 수집 중 발생한 예외의 표현 등을 처리한다.

UIFS 컴포넌트는 UI조각들을 병렬로 수집하여 수집된 순서대로 Reverse AJAX 기법으로 CA의 UICS 호출을 통해 UI조각을 UI레이아웃과 조합한다. UIFS 컴포넌트에서 UI조각의 수집은 스레드 풀을 사용한다. 스레드 풀은 스레드를 미리 생성하여 풀로 관리하기 때문에 필요할 때 마다 생성하는 것에 비해 성능이 향상되지만 너무 많은 스레드를 관리하는 것은 오히려 성능저하에 원인이 된다. 따라서 UIFS 컴포넌트에는 이 스레드 풀을 관리하기 위한 몇 가지 설정이 존재하고 시스템 별로 이 설정값을 조정하여 최적화된 성능을 제공한다.

CM는 각 UI와 UI에 포함되는 UI조각의 정보를 관리하는 컴포넌트이다. 이 컴포넌트가 관리하는 정보는 권한에 관한 정보를 포함한다. 즉 사용자가 가지는 권한에 따라 접근 가능한UI 및 UI조각으로 화면 정보를 재구성하여 PLS 컴포넌트 및 UIFS컴포넌트에 제공한다.

5.2 구현 결과

다음 그림은 UI 페이지를 구성한 예시 화면이다.



(그림 6) UI 페이지 예시
(Figure 6) UI-Page Example

예시 화면은 12개의 UI조각으로 구성된 UI 페이지이다. 각각의 UI조각은 서로 다른 서비스 제공자에게 원격 호출을 통해 정보를 수집되며 수집된 순서대로 화면에 표현된다.

5.3 성능평가

본 연구에서는 제안한 매쉬업 아키텍처의 성능을 검증하기 위해 기존의 방식인 서버 기반 매쉬업과 클라이언트 기반 매쉬업 그리고 제안한 매쉬업 아키텍처 간의 성능을 비교 평가하였다.

성능평가는 반응시간과 응답시간 측정을 위해 한 사용자가 접근하였을 때의 수치를 측정하였고, 부하 상태에서 시스템의 처리량과 응답시간을 측정하기 위해 JMeter를 이용하여 부하테스트를 진행하였다.

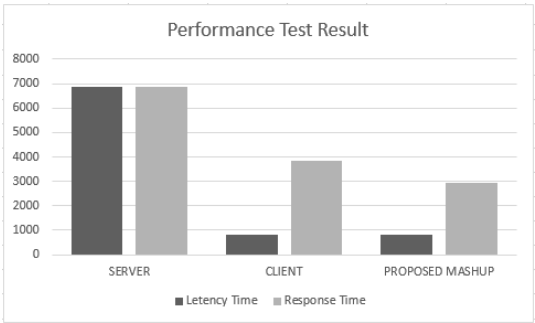
측정은 16개의 UI조각을 서비스하는 화면으로 모든 UI조각이 원격 서비스 호출을 통해 수집되는 화면을 대상으로 서비스 방식만을 달리하여 각각의 수치를 측정하였다.

성능평가 환경은 다음과 같다.

(표 1) 성능평가 환경
(Table 1) The Performance Test Environment

	서버 환경	클라이언트 환경
CPU	Intel Xeon Quad Core 2G, 2CPU	Intel Core i5 2.3G
Memory	8 G	4 G
OS	Windows 2008 64Bit	Windows 7 64 Bit
Tool	Tomcat 7.0	JMeter 2.9

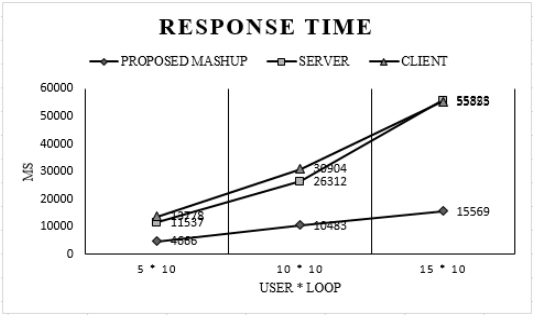
반응시간 및 응답시간은 서버로 UI 페이지를 요청한 순간부터 각각의 아키텍처가 최초의 응답을 전송한 시간을 반응시간으로 마지막 패킷이 수신된 시간까지를 응답시간으로 측정하였으며 결과는 다음과 같다.



(그림 7) 성능평가 결과
(Figure 7) Performance Test Result

성능측정 결과 제안한 아키텍처는 서버 기반 매쉬업에 비해 약 11%, 클라이언트 기반 매쉬업과는 비슷한 반응시간을 기록하였다. 응답시간 측정에서 제안한 아키텍처는 서버 기반 매쉬업에 비해 약 43%, 클라이언트 기반 매쉬업에 비해 약 77%의 시간만 소모하는 것을 확인할 수 있었다.

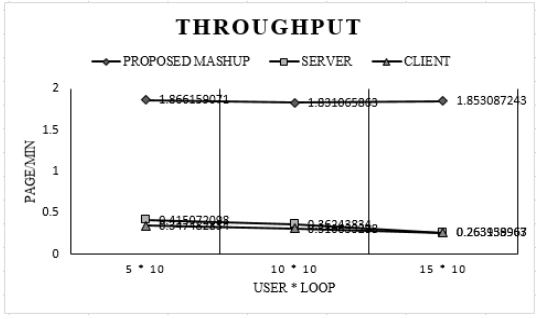
부하 테스트는 JMeter를 사용하여 가상 사용자 수를 5, 10, 15 로 5씩 증가시켜 동일한 페이지를 10번씩 호출하여 응답시간의 평균과 처리량을 측정하였다. 측정 결과 응답시간은 다음과 같았다.



(그림 8) 응답시간
(Figure 8) Response-Time

응답시간 측정 결과 서버 기반 매쉬업이나 클라이언트 기반 매쉬업의 경우 가상 사용자 수를 증가 함에 따라 응답 속도에 큰 변화를 보였고 제안한 아키텍처에 비해 2 ~ 3배 이상의 응답시간을 기록하였다.

처리량 측정결과는 다음과 같다.



(그림 9) 분당 처리량
(Figure 9) Throughput per Minute

처리량 측정 결과 서버기반 매쉬업과 클라이언트 기반 매쉬업은 0.5 ~ 0.2 사이의 분당 페이지 처리량을 보인 반면 제안한 아키텍처는 타 매쉬업 아키텍처에 비해 4배 이상의월등한 처리량을 보였다.

6. 결 론

서비스가 다양화, 분산화 되고 있는 현재의 웹에서 UI-Mashup의 활용도는 날로 증가하고 있다. 하지만 UI-Mashup 관련 기존의 연구는 자동화된 매쉬업 기법에 대한 연구가 주로 이루어지고 있고 성능에 대한 연구들은 현재의 사용자 환경을 따라가지 못하고 있다. 따라서 사용자에게 UI-Mashup은 여전히 느리고 불편하다.

본 연구는 UI-Mashup의 느린 서비스 문제를 해결하기 위한UI-Mashup 아키텍처를 제안한다. 제안한 아키텍처는 UI레이아웃과 UI조각의 수집 및 UI-Mashup의 과정을 분리하여 반응시간을 향상시켰고 매쉬업 화면을 구성하는 UI조각의 수집을 서버에서 처리하여 보안 및 안정성을 제공하였다. 또한 UI조각의 수집을 병렬로 처리하여 전체적인 응답시간을 향상 시켰으며, 수집된 UI조각을 별도의 전송 채널을 통해 수집된 순서대로 클라이언트로 전송하여 매쉬업 시간을 단축 시켰다.

성능 평가 결과 제안한 아키텍처는 기존의 서버기반 매쉬업과 클라이언트 기반 매쉬업에 비해 빠른 응답시간과 월등히 높은 처리량을 제공할 수 있었다.

본 연구의 결과는 포털 시스템 또는 대시보드와 같이 분산된 UI조각들을 모아 한 화면에서 서비스하는 소프트웨어에 적용될 경우 높은 성능 향상을 제공할 수 있다. 또한 UI레이아웃의 제공 및 UI조각의 처리가 모두 서버에서 수행되기 때문에 QoS 관리 및 보안적인 제어에도 용이하게 적용될 수 있다.

참 고 문 헌(Reference)

[1] Jian Meng; Jinlong Chen, "A Mashup Model for Distributed Data Integration," Management of e-Commerce and e-Government, pp.168-171, Sept. 2009.

[2] Azar, S.N., jalali, A.H., Falsafi, S., "Commercial Portals Evaluation," Advanced Communication Technology, ICACT 2008. 10th International Conference on, vol.3, pp.2035-2039, Feb. 2008.

[3] Stefan Hepper, "JavaTM Portlet Specification Version 2.0", pp. 33-36, IBM Corp, 2008.

[4] Gang Huang, Qi Zhao, Jiyu Huang, Xuanzhe Liu; Teng Teng, Yong Zhang, Honggang Yuan, "Towards service composition middleware embedded in web browser", CyberC '09. International Conference on, pp.93-100, Oct. 2009.

[5] Minhas, S.S., Sampaio, P., Mehandjiev, N., "A Framework for the Evaluation of Mashup Tools," Services Computing (SCC), 2012 IEEE Ninth International Conference on, pp.431-438, June. 2012.

[6] "IBM Mashup Center Release 2.0 Performance Tuning Guide", pp.17-19. IBM Corp, 2010.

[7] "IBM Mashup Center Release 2.0 Administering IBM Mashup Center", pp.41. IBM Corp, 2010.

[8] Lee, J., Ying-Yan Lin, Shang-Pin Ma, Yao-Chiang Wang, Shin-Jie Lee, "Integrating Service Composition Flow with User Interactions", Service-Oriented System Engineering '08. IEEE International Symposium on, pp.103-108, Dec. 2008.

[9] Martin Fowler, "Patterns of Enterprise Application Architecture", pp.16-17, 2002.

[10] Laga, N.,Bertin, E., Crespi, N., "Composition at the frontend: The user centric approach", Intelligence in Next Generation Networks 14th International Conference on, pp.1-6, Oct. 2010.

[11] Dave Crane, Phil McCarthy, "Comet and Reverse Ajax The Next-Generation Ajax 2.0", pp.1-2, 2008.

[12] "Same Origin Policy", http://www.w3.org/Security/wiki/Same_Origin_Policy, W3C, 2010.

[13] Latih, R., Patel, A.M., Zin, A.M., Yiqi, T.,Muhammad, S.H., "Whip: A framework for mashup development with block-based development approach",International Conference on Electrical Engineering and Informatics, pp.1-6, July. 2011.

[14] Incheon Paik, Wuhui Chen, Michael N. Huhns, "A Scalable Architecture for Automatic Service Composition," IEEE Transactions on Services Computing, pp.1-14 , Nov. 2012.

[15] Qi Zhao, Gang Huang, Jiyu Huang, Xuanzhe Liu,Hong Mei, "A Web-Based Mashup Environment for On-the-Fly Service Composition," Service-Oriented System Engineering IEEE International Symposium on, pp.32-37, Dec. 2008.

[16] Joe McKendrick, "AJAX and enterprise 2.0 for that 'last mile' of SOA," unpublished.

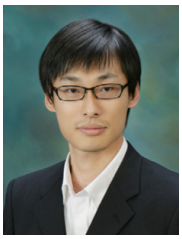
[17] Xuanzhe Liu, Yi Hui, Wei Sun, Haiqi Liang, "Towards Service Composition Based on Mashup", 2007 IEEE Congress on Services, 2007.

[18] Arto Salminen, Tommi Mikkonen, Feetu Nyrhinen, Antero Taivalsaari, "Developing client-side mashups: experiences, guidelines and the road ahead", In Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek '10), pp 161-168, ACM New York, 2010.

[19] Pietschmann S, Voigt M, Meissner K, "Dynamic Composition of Service-Oriented Web User Interfaces", ICIW '09. Fourth International Conference on, pp.217-222, May 2009.

[20] Cappiello C, Matera M, Picozzi M, Daniel F, Fernandez A, "Quality-Aware Mashup Composition: Issues, Techniques and Tools", Quality of Information and Communications Technology (QUATIC) 2012 Eighth International Conference on, pp.10-19, Sept 2012.

◐ 저 자 소 개 ◑



조 동 일(Dong-Il Cho)

2003년 수원대학교 기계공학과 학사
2008년 숭실대학교 정보과학 대학원 소프트웨어공학과 석사
2012년 숭실대학교 컴퓨터학과 박사
2003년~현재 (주)토마토시스템 기술연구소 책임연구원
관심분야 : 웹 애플리케이션 아키텍처, 소프트웨어 아키텍처, 미들웨어