DOI: 10.3745/KIPSTD.2011.18D.3.205

클라이언트 매시업의 편리한 이동 사용자 인터페이스 개발

이 은 정†

요 약

클라이언트 기반 웹 매시업은 웹 어플리케이션 개발의 주요 아키텍처로 자리잡았다. 웹서비스에 대한 프리젠테이션 코드를 자동생성하는 방법은 잘 알려져 있으나 뷰와 서비스 요청 간의 이동과 사용자 인터페이스 통합부는 대부분 수동으로 개발되고 있다. 본 논문에서는 한 서비스 요청의 결과 데이터로부터 다른 서비스 요청의 입력 매개변수를 연결하는 데이터 바인딩을 정의하고 이를 바탕으로 매시업의 편리한 이동 사용자 인터페이스 개발 방법을 소개한다. 주어진 서비스 집합과 데이터 매핑에 대해 컨텍스트 기반의 바인딩을 찾고 그를 기반으로 뷰와 이동 메뉴를 생성하는 방법을 소개한다. 제안된 방법은 서비스의 개수와 매시업 페이지의 복잡도가 높아지는 경우에도 사용자들이 클라이언트 매시업 페이지의 서비스 간에 이동하기 위한 간편하고 효과적인 이동 메뉴를 제공할 수 있다. 제안된 방법의 유용성을 보이기 위해 관광 안내 서비스를 보인다.

키워드 : 서비스 조합, 매시업, 데이터 매핑, 사용자 인터페이스

Development of Efficient User Navigation Interface for Client-side Mashups

Eunjung Lee†

ABSTRACT

Client based web mashups have become one of the most important architecture in web application development. Although there are well known methods to generate presentation view codes for web services, navigations between views and service requests are still developed manually in most web mashups. In this paper, we propose the concept of deterministic data binding from output data to input parameters of another method. Using binding relations, we can model navigation menus for service method requests. For a given set of data mappings between services, we investigate context dependent binding conditions and discuss the generation of views and navigation menus. The proposed approach provides UI for users to navigate services of client mashup page using simple and convenient interface even when the number of services and the size of the mashup page grows. In order to show the usability of the proposed approach, we present a historic tourism service.

Keywords : Service Composition, Mashup, Data Mapping, User Interface

1. 서 론

서비스 조합과 매시업은 웹 어플리케이션 개발을 위한 핵심 기술로 자리잡았다. 또한 웹서비스의 형태로 제공되는 컨텐츠와 기능이 점점 많아지고 서비스들이 장소나 컨텍스트 등에 따라 동적으로 변하면서 이러한 환경에 더 잘 적응할 수 있는 사용자 중심의 클라이언트 측 매시업이 주목을 받고 있다. 그런가하면 하나의 클라이언트 매시업 페이지가 인터페이스하는 서비스나 자원이 점점 더 많아지고 복잡해지면서 체계적인 개발 방법과 코드 생성 기술의 필요성이 높아지고 있다.

동적인 서비스 환경을 지원하기 위해서는 주어진 서비스 집합에 대해 동적으로 클라이언트 코드가 자동 생성되어야 한다. 한편 클라이언트 페이지가 인터페이스하는 서비스의 개수가 증가하고 매시업 관계도 복잡해지면서 클라이언트 페이지 내에서의 이동을 설계하는 것이 큰 문제로 등장하였다. 매시업 페이지의 이동 코드를 생성하기 위해서는 결과 데이터로부터 가능한 서비스 조합 관계를 찾고, 서비스 호출을 위해 전달할 매개변수를 결정할 수 있어야 한다. 또한 복잡한 서비스 간의 관계를 사용자가 간편하게 제어할 수 있도록 사용자의 상호작용을 최소화한 이동 및 제어 코드의 생성이 가능해야 한다.

※ 이 논문은 2010년 경기대학교 연구년 수혜로 연구되었음.
† 종신회원 : 경기대학교 컴퓨터과학과 부교수
논문접수 : 2011년 4월 7일
수 정 일 : 1차 2011년 5월 20일
심사완료 : 2011년 5월 21일

매시업 사용자 인터페이스 개발은 매우 활발하게 연구되고 있으며 [1,2], 최근 동적이고 다양한 서비스 환경을 지원하기 위해 클라이언트 측 매시업 개발을 위한 사용자 인터페이스 통합이 관심을 끌고 있다[3-5]. 그러나 데이터 흐름과 서비스 조합의 연구 결과를 이동 사용자 인터페이스 개발 방법에 적용할 수 있는 방법은 제시된 바가 없다. Lecue 등이 서버에서의 서비스 조합을 위해 데이터 흐름을 자동화하는 방법을 제시하였으나[6] 저자들이 조사한 바로는 클라이언트에서 컨텍스트를 이용한 이동과 데이터 전달 문제를 다룬 연구는 없는 것으로 보인다.

본 논문에서는 입력 매개변수로 전달할 데이터의 결정을 데이터 바인딩이라고 부르는데, 이것은 서비스 조합을 지원하는 클라이언트 개발을 위해 해결되어야 하는 주요한 문제이다. 특히 결정적인 바인딩은 사용자의 추가적인 상호작용 없이 다음 요청에 필요한 데이터를 결정할 수 있는 경우이다. 또한 반복되는 데이터 부분에서 현재의 컨텍스트를 이용하여 매개변수를 결정할 수 있는 반복부 바인딩 개념을 소개한다.

제안된 방법을 이용하여 Ajax 기반의 관광 안내 서비스 페이지를 생성한다. 관광 안내 서비스는 박물관 서비스와 관광 서비스로 구성되는데 자동생성된 사례 매시업 페이지를 통해 결과 코드가 컨텍스트 기반으로 매시업 관계를 편리한 메뉴로 제공하고, 사용자의 상호작용을 최소화할 수 있음을 보인다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 소개하고 3장에서 제안된 데이터 매핑 모델과 반복부 바인딩의 개념을 소개한다. 4장에서는 코드 생성 시스템의 구현을 알아보고 관광 안내 서비스 사례의 생성된 코드를 소개한다. 6장에서는 결론을 맺는다.

2. 배 경

서비스 조합은 서비스 간에 데이터를 전달하여 연결하는 방법이다. 매시업과 서비스 통합은 최근 많은 연구가 진행되었다. 본 논문에서는 동적 서비스 환경에서 서비스 통합과 클라이언트 페이지 내의 이동을 위주로 하는 사용자 인터페이스 개발 방법을 살펴본다.

최근에 동적 서비스 환경에서 서비스의 발견, 선택, 호출, 조합 기술이 많은 관심을 끌고 있다. Li 등은 무선모바일 네트워크 환경에서 서비스의 생성, 변경 등을 모니터링하는 디스페처 시스템을 두고 동적으로 서비스하는 프레임워크를 발표하였다[7]. 또한 서비스의 조합 뿐 아니라 사용자 맞춤형의 추천 기능을 연구한 사례도 있었다[8]. 이러한 동적 서비스 환경에서 서비스의 조합이 가능하려면 사용자에게 필요한 인터페이스를 제공하는 클라이언트 코드의 자동 생성이 필요하다.

광범위하게 보급되고 있는 웹서비스를 활용하기 위하여 매시업 사용자 인터페이스 개발은 매우 활발하게 연구되고 있다[1, 2]. 서버 측 매시업 연구는 서비스 간 데이터 흐름을 분석하고 조합하는 방법을 위주로 하였다면[9-11], 클라이언트 측 매시업은 프리젠테이션 계층의 통합이 주로 다루어졌다[3,12,13]. 최근 동적이고 다양한 서비스 환경을 지원하기 위해 클라이언트 측 매시업 개발이 주목을 받고 있다[3-5]. 클라이언트 측 매시업은 서비스 검색, 조합, 제어 등을 모두 서버에서 수행하는 모델로 서비스 통합을 위한 어플리케이션, 데이터, 사용자 인터페이스 계층을 모두 포함하는 개발 프레임워크가 활발하게 연구되고 있다[2,4,5]. 그런가 하면 서비스 조합을 이동의 관점에서 다루는 연구도 있었다[14].

MashArt는 컴포넌트 기반의 매시업 개발 도구로 어플리케이션, 데이터, 프리젠테이션의 세 층을 모두 통합할 수 있는 모델을 제시하였다[4]. Pietschman 등은 CRUISe 프레임워크를 제안하였는데, 이것은 일명 “thin server architecture”를 도입하여 매시업을 개발하기 위한 컴포넌트 기반의 도구와 개발 환경을 제공한다[5]. Nestler 등은 서비스 조합의 사용자 인터페이스 편집기를 포함한 모델 기반의 개발 방법을 바탕으로[12] ServFace 프레임워크를 개발하였다. 어플리케이션과 프리젠테이션 층의 통합의 관점에서 이들 프레임워크를 비교한 연구도 있었다[13]. 이상의 프레임워크들은 다양한 웹서비스 프로토콜과 데이터 포맷을 지원하며 모델 기반의 개발 방법을 적용하여 자동생성을 위해 큰 규모의 라이브러리를 사용한다. 이와 비교하여 본 논문에서는 가벼운 웹서비스 프로토콜인 REST 방식을 택하였으며, 무거운 프레임워크가 아니라 어디서든 사용할 수 있는 간결한 HTML과 JavaScript 코드를 생성하고자 한다.

그런가하면, 저자들의 이전 논문에서는 클라이언트 측 매시업 개발을 위해 RESTful 서비스 패턴을 이용한 이동 설계를 제안하였고 XForms 기반의 코드 생성 방법을 제안하였다[14,15]. 본 논문에서는 서비스 패턴에 대한 가정 없이 서비스 메소드의 타입 분석을 통해 컨텍스트 메뉴의 자동생성과 매개변수 바인딩을 계산하는 방법을 다룬다.

매시업 클라이언트 페이지에서 다루는 서비스의 수가 커지면서 뷰와 서비스 요청 및 회신 간의 관계가 매우 복잡해질 수 있다. 이러한 문제는 Pietschman 등이 제안한 프레임워크 위주의 접근 방법에서는 컴포넌트 들 간의 조합(composition logic)의 일부로 취급되었다[4,5]. 그런가 하면 Lecue 등은 데이터 조합의 유형을 분석하여 서버에서 데이터 흐름을 자동화하는 어댑터 모듈을 제안하였다[6]. 또한 Dutch 등은 복잡합 매시업 어플리케이션에서의 이동 모델을 제시하였다[16]. 본 논문은 데이터 흐름을 위한 바인딩 관계를 찾고 컨텍스트 메뉴를 동적으로 자동생성하는 방법을 제시하고자 한다. 이것은 Lecue 등이 제시한 서버에서의 데이터 흐름 어댑터 모듈을 클라이언트 페이지 내에서 이동의 관점에서 접근한 것으로 볼 수 있다. 그러므로 본 논문에서는 효율적인 개발의 측면 뿐 아니라 결과 코드가 사용자에게 유용한 이동 인터페이스를 제공하는 것도 중요한 목표가 된다.

3. 뷰와 이동 메뉴의 유형

뷰와 서비스 간 이동을 설계하는 기본 정보는 서비스 메소드의 원형에 정의된 입력 및 결과 데이터 타입이다. 그리고 이동 메뉴가 사용자 주도 매시업 제어의 설계 결과를 표현한다.

보통 서비스 메소드는 타입, URL, 메소드 원형으로 정의된다. 주어진 메소드 m에 대해 메소드 원형은 결과 데이터 타입 $output(m)$과 입력 매개변수 타입의 집합 $input(m)$으로 표시된다. 여기서는 타입 기반의 서비스 조합을 위해 모든 입력 매개변수가 요소명 기반 타입을 가진다고 가정한다.

C가 다루어지는 모든 서비스 메소드 집합이라 하자. 클라이언트 페이지는 메소드 $m \in C$에 대해 입력뷰 $view_{in}(m)$이나 결과뷰 $view_{out}(m)$을 가질 수 있다. 입력뷰는 입력 매개변수를 편집하고 제출하기 위한 뷰이고 결과 뷰는 사용자가 결과 데이터를 확인하는 뷰이다. 뷰의 배치와 디스플레이 관계는 본 논문의 범위를 벗어나므로 여기서는 (그림 1)과 같이 하나의 뷰가 스크린 전체를 차지한다고 가정한다.

다음으로 뷰와 서비스 요청의 관계를 살펴본다. 여기서는 요청 유형을 요청 메뉴 형태와 전달할 데이터를 결정하기 위한 사용자 상호작용을 중심으로 분류한다. m_1, m_2, m_3가 C에 속하는 메소드라 할 때, 다음과 같은 네 가지 이동 유형을 얻을 수 있다. 이들은 (그림 1)의 (a)~(d)의 흐름 관계에 대응한다.

(a) 사용자는 어느 곳에서든 이동할 수 있는 뷰 이동 요청 또는 입력 매개변수가 없는 메소드 호출 요청

(b) 사용자는 $view_{in}(m_2)$ 에서 m_2 메소드를 요청할 수 있다. (submit버튼)

(c) 사용자는 $out(m_1)$ 이 $input(m_2)$ 를 제공할 수 있다면 $view_{out}(m_1)$ 에서 m_2를 요청할 수 있다.

(d) 사용자는 $view_{out}(m_1)$ 에서 다음과 같은 조건이 만족하면 m_3를 직접 요청할 수 있다.
i) $output(m_1)$ 에서 $input(m_2)$ 를 제공할 수 있으며,
ii) $output(m_2)$ 가 $input(m_3)$ 를 제공할 수 있으며,
iii) 사용자의 개입 없이 m_2의 회신 핸들러에서 m_3를 호출할 수 있다.

서브미트 버튼으로 입력 뷰에서 해당 서비스를 요청하는 것은 웹서비스 인터페이스에서 일반적인 유형이다. 그러나 결과 데이터를 다음 요청 메소드의 입력데이터로 바로 연결하는 (c) 유형이나 또 다른 메소드를 연결하여 호출하는 (d) 유형은 매시업 서비스의 요청이다. 이것은 m_1의 결과뷰 데이터로부터 다음 요청할 메소드 m_2의 입력매개변수를 제공할 수 있는 경우로, 이를 본 논문에서는 바인딩이라고 부르고, $m_1 \rightarrow m_2$로 표시한다.

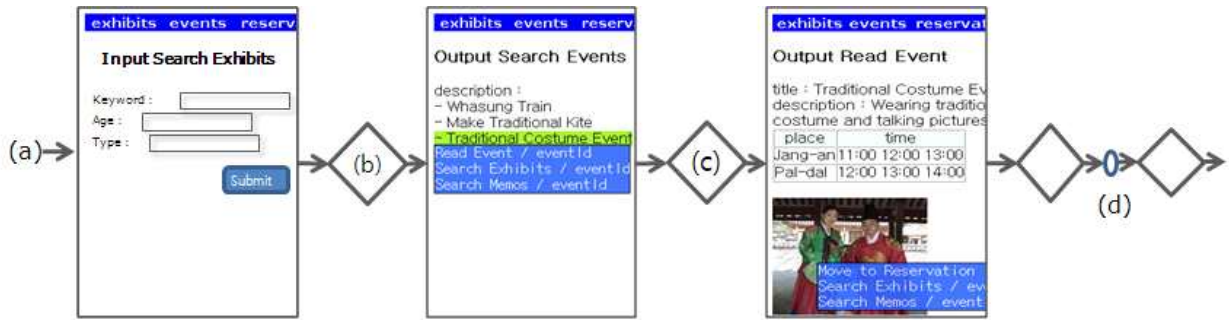
메소드 간의 바인딩 관계는 메소드 수가 증가하고 매시업 페이지가 커지면 상당히 복잡해 질 수 있다. 유용한 사용자 인터페이스를 제공하기 위해서는 적절한 이동 메뉴의 선택과 배치가 중요한 문제이다. 자동으로 다음 요청에 대한 입력 매개변수의 바인딩이 가능하다면 사용자 상호작용을 줄일 수 있다.

(그림 1)과 같은 이동 유형에 따라, 이동 메뉴를 분류해 볼 수 있다. 첫 번째 유형은 정적 메뉴로, 어느 뷰에서든 접근 가능한 메뉴이며 (그림 2(a))와 같이 네비게이션 바의 메뉴 형태로 제공될 수 있다. 다음 유형은 컨텍스트 메뉴인데, 매개변수 바인딩을 제공하는 결과뷰에 내장된다. 컨텍스트 메뉴는 (그림 2(b))와 같은 뷰메뉴, 또는 (그림 2(c))와 같은 팝업 메뉴 형태로 제공된다. 이를 좀더 구체적으로 분류해 보면 다음과 같다.

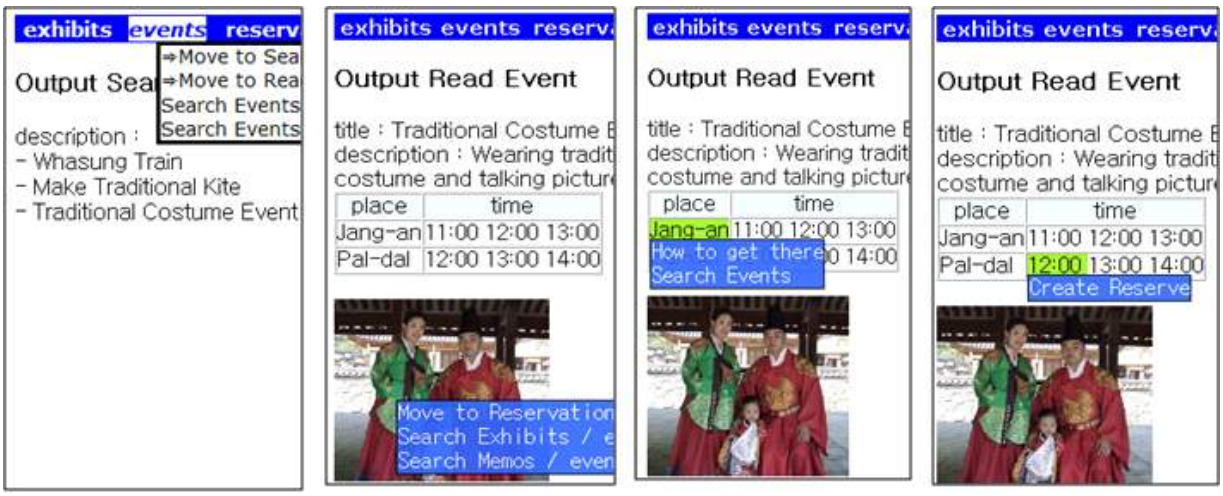
(1) 정적 메뉴 유형
- 뷰 이동 메뉴 : 입력 뷰나 이전에 수신한 결과 뷰로 이동하는 메뉴
  (예 : 그림 2(a)의 => Move to Search Output)
- 매개변수 없는 요청 전송 : 매개변수 전달 없이 요청할 수 있는 호출 메뉴
  (예 : 그림 2(a)의 Search Events)

(2) 컨텍스트 메뉴
- 뷰 메뉴 : 현재의 결과 데이터로부터 결정적으로 데이터 바인딩이 가능한 요청 메뉴
  (예 : 그림 2(b)의 Search Exhibits with eventId)
- 팝업 메뉴 : 현재 뷰의 해당 컨텍스트에 의해 데이터 바인딩이 가능한 요청 메뉴
  (예 : 그림 2(c)의 Search Events with place)

이렇게 유형을 나누는 것은 매시업에서 이동을 위한 사용자 인터페이스 메뉴가 뷰의 필요한 곳에 포함되어 전체적으



(그림 1) 전달 데이터와 결과 처리 방식에 따른 서비스 요청 유형 분류



(a) Static menus (b) View menus (c) Context popup menus

(그림 2) 메뉴 유형에 따른 배치 사례

로 메뉴의 복잡도를 줄일 수 있고 사용자가 컨텍스트 기반으로 매시업 흐름을 선택할 수 있다는 장점을 가진다.

4. 매개변수 바인딩과 바인딩 컨텍스트

이 장에서는 주어진 메소드 m_1, $m_2 \in C$에 대해 $m_1 \rightarrow m_2$인 경우 m_2의 입력 매개변수로 전달될 데이터 바인딩을 찾는 것을 목표로 한다. 가능한 서비스 조합을 구하기 위해 서비스 명세와 함께 데이터 매핑 관계가 주어졌다고 가정한다.

데이터 매핑은 서비스 간 또는 도메인 간의 데이터 타입을 연결시키는 관계 집합이다. 본 논문에서는 타입 기반의 데이터 매핑으로 타입이 동일하거나 사용자 지정 매핑에 의해 명시된 경우 매핑 가능하다고 본다. 주어진 서비스 집합 C에 대해 매핑 M은 M={(t_1, t_2) | $t_1 \neq t_2$, 사용자가 정의한 매핑 관계}로 정의되며, 타입 t에 대해 $mappable_M(t)$ = {t' | (t', t) ∈ M}이다.

메소드 간의 데이터 전달을 분석하기 위해 Σ = $output(m_1)$이고 $input(m_2)$ = [p_1, p_2, ..., p_k]라 하자. 주어진 타입 t에 대해 Σ로부터 바인딩될 수 있는 요소의 경로 집합을 $bindings(\Sigma, t)$ = {π / $\pi(\Sigma)$의 노드가 $mappable(t)$에 속하는 타입을 가짐}. 그러면 입력 매개변수 리스트에 대한 바인딩은 다음과 같이 정의된다.

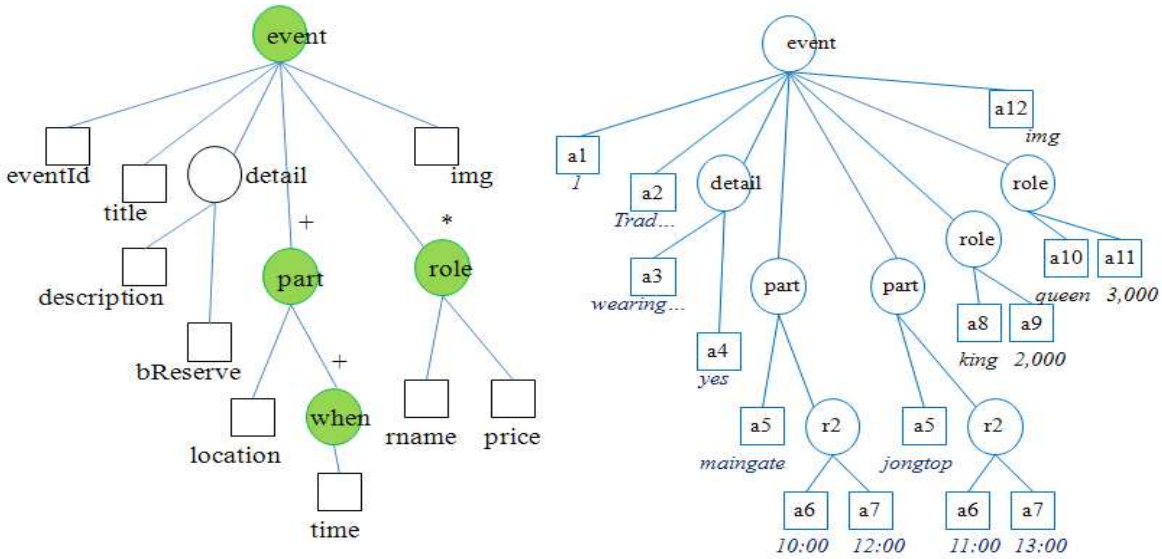
$bindings(\Sigma, [p_1, p_2, ..., p_k])$ = {$[\pi_1, \pi_2, ..., \pi_k]$ / $\pi_i \in bindings(\Sigma, p_i)$ for $1 \leq i \leq k$ }.

그러므로 $m_1 \rightarrow m_2$ 관계는 $bindings(output(m_1), input(m_2))$가 공집합이 아닐 때 만족한다. 한편 트리 타입 Σ에 대해 bindings(Σ, input(m_2))가 유일하면 Σ d_binds m_2라고 한다. 그리고 m_1의 결과데이터 타입 Σ에 대해 Σ d_binds m_2면 이를 결정적 바인딩 관계라고 하고 $m_1 \Rightarrow m_2$로 표시한다. $m_1 \Rightarrow m_2$가 만족하는 경우에는 m_2의 입력매개변수로 전달되어야 할 데이터를 사용자 개입 없이 m_1의 결과에서 결정할 수 있다. 예를 들어 (그림 3(a))의 트리 타입 Σ_1은 m_2 = "Search Exhibits Related to the Event", $input(m_2)$ = {eventId}에 대해 트리 타입 Σ_1에서 eventId이 유일하게 등장하므로 m_2를 결정적으로 바인딩한다. 즉 Σ_1 d_binds m_2이다.

이하에서는 결정적 바인딩 조건을 스키마 정의 타입으로부터 정적으로 계산하는 방법을 살펴본다. 흔히 반복되는 부분트리가 정해지면 매개변수의 결정적 바인딩이 가능한 경우가 많은데, 이러한 경우를 처리하기 위하여 트리 타입의 반복 구조를 분석하여 결정적 바인딩이 가능한 컨텍스트를 찾는다.

이하에서 사용되는 반복부 트리 및 반복부 노드의 개념은 [15]에서 소개되었다. 반복부 노드는 트리 구조 정의에서 최대 등장회수가 1보다 큰 요소로 정의된다. (그림 3(a))에는 part, when, role의 세 개의 반복부 노드가 있다. 반복되지 않는 후손을 직접 후손이라 한다. 그 중에서도 텍스트 또는 속성 데이터를 가지는 터미널 노드만 모은 집합을 터미널 직접후손이라 하는데, (그림 3)에서 터미널직접후손(event) = {eventId, title, description, bReserve, img}이다.

Σ = $output(m_1)$이고 r이 Σ의 반복부 노드라 하자. 노드 r의 Σ에 대한 클로저 부분트리는 $closure(\Sigma, r)$로 표시되고 모든 조상 반복부 노드와 그들의 직접후손, 그리고 r을 루트로 하는 반복되지 않는 부분 트리를 포함한다. 즉 r의 형제와 조상의 다른 반복부에 속하는 부분을 제외한 전체 트리를 클로저 부분트리로 정의한다. 예를 들어 part의 클로저 부분트리는 event와 직접 후손, part의 부분트리를 포함한다. 또한 rname의 클로저 부분트리는 event의 직접 후손과 role의 부분트리를 포함하며, 이것은 (그림 3(b))의 트리 인스턴스에서 점선으로 표시된 부분이 된다.



(a) output(ReadEvent)의 타입 정의 Σ1 (b) ReadEvent 요청의 결과 트리 인스턴스

(그림 3) event 결과 데이터의 타입 정의부 및 트리 인스턴스

주어진 반복부 클로저 부분트리 *closure(Σ, r)*에서 *input*(m_2) 의 모든 매개변수에 대해 유일한 바인딩을 찾을 수 있다면 *closure(Σ, r)*이 m_2를 결정적으로 바인딩한다. 예를 들어 (그림 2)에서 *closure*(Σ, part) *d_binds* [eventId, location]이고 *closure*(Σ, a10) *d_binds* [eventId, location, time]을 만족한다.

주어진 반복부 노드 *r*에 대해 이 노드의 클로저 트리가 *d_bind* 할 수 있는 메소드들을 찾는 것이 우리의 관심이다. 이를 다음과 같이 메소드 쌍에 대한 바인딩 컨텍스트로 정의한다.

정의 1. 주어진 메소드 m_1, m_2에 대해 Σ = *output*(m_1)이고 *r*은 Σ의 반복노드라 하자. 그러면 *r*의 바인딩 컨텍스트는 다음과 같이 정의된다.

bindingContext(m_1, m_2) = {(*r*, [π_1, π_2, ..., π_k]) | *input*(m_2) = [p_1, p_2, ..., p_k], $m_2 \in C$, π_r이 Σ에서 r의 경로라 할 때 다음 조건을 만족한다.

i) *bindings*(*closure*(Σ, *r*), p_i) = {π_i'}, where $\pi_r.\pi_i = \pi_i'$, $1 \le i \le k$, and

ii) *r*의 조상 중에는 위의 조건을 만족하는 반복부 노드가 없음.

}

i)의 조건은 *r*의 반복부 클로저 트리에서 p_i에 바인딩될 수 있는 요소가 유일함을 나타내고 π_i는 *r*로부터의 그 요소까지의 상대 경로이다. ii)의 조건은 반복부 노드 *r*이 m_2를 결정적으로 바인딩하는 첫 번째 노드임을 나타낸다.

(그림 2)의 사례 예제에서 Σ = *output*(ReadEvent)가 (그림 3(a))에 나타나 있다. 이 때 *input*(CreateReserve3) = [eventId, rname]이라고 하면 role 노드가 이 메소드를 결정적 바인딩하는 반복부 노드가 된다. 한편 *input*(CreateReserve2) = [eventId, location, time]이라면 location과 time을 결정해야 하므로 when 노드가 이 메소드를 결정적 바인딩하는 반복부 노드이다. 그러므로 다음과 같은 바인딩 컨텍스트를 얻을 수 있다.

(/event/role,[../eventId,./rname]) ∈ bindingContext(ReadEvent, CreateReserve3)

(/event/part/when, [../../eventId, ../location, ./time]) ∈ bindingContext(ReadEvent, CreateReserve2)

[알고리즘] 바인딩 컨텍스트 계산 알고리즘

입력 : *M* : 데이터 매핑, 메소드 m_1, $m_2 \in C$, $[p_1, p_2, ..., p_k]$: m_2의 매개변수 리스트, Σ : *output*(m_1)

결과 : 반복커버가 만족되면 *RepeatBinding*(m_1, m_2) = (π_R, [π_1, π_2, ..., π_k])를 계산함

1. $\pi_R \leftarrow$ *null*, $\pi_i \leftarrow \varepsilon$.
2. $\forall i$, $1 \le i \le k$:
   2.1 i번째 매개변수에 대한 반복노드 *RList[i]* ← *null*,
   2.2 i번째 매개변수에 대한 상대경로 *RelPathList[i]* ← ε.
3. Σ의 루트 반복노드 R_{root}에 대해
   3.1 *check_repeat_cover*(R_{root}) 호출
   3.2 결과가 거짓이면 매핑이 존재하지 않음. 종료.
4. 결과가 참이면,
   4.1 *RList[1..k]*에서 최하위 노드가 반복기준노드가 됨
   4.2 $\pi_R \leftarrow$ 반복기준노드의 절대 경로

4.3 $\forall i, 1 \le i \le k$:
4.3.1 $\pi_i \leftarrow$ *RList[i]*의 π_R에 대한 상대 경로 + *RelPathList[i].*

procedure **check_repeat_cover(r)**
1. $\forall i, 1 \le i \le k$:
   1.1 *count[i]* ← *r*의 직접자손 중 *mappable*(p_i)에 속하는 타입 요소의 등장회수
   1.2 *count[i]* > 1이면 *return false; // R*의 직접 자손에서 다중 등장
2. $\forall i, 1 \le i \le k$ s.t. *count[i]* = 1 :
   2.1 *RList[i]* ≠ *null*이면 *return false;* // 조상에서 이미 바인딩된 경우, 바인딩 충돌
   2.2 *RList[i]* = *r; // 현재 노드를 i*번째 바인딩으로 등록함
   2.3 *relPathList[i]* ← 바인딩된 요소의 상대경로
   2.4 해당 타입 요소가 생략 가능하면 정적 바인딩 ← *false*
3. if *RList[j]* ≠ *null* $\forall i, 1 \le i \le k$, *return true.*
4. $\forall r' \in children(r)$: *check_repeat_cover(r')* 호출
4. 결과가 거짓이면 // 자손에서 바인딩에 실패함 : 다중 등장 또는 바인딩 충돌
   4.1 $\forall i, 1 \le i \le k$: *RList[i]* ← *null; // 바인딩을 리셋함*
   4.2 *return false;*
6. *return true.*

5. 구현 결과

4장에서 소개한 바인딩 컨텍스트는 컨텍스트 메뉴를 생성하기 위한 정보를 제공하며, 본 논문에서는 자바스크립트 코드 생성 시스템 MMMash에 확장하여 제안된 방법을 적용하는 프로토타입을 개발하였다. 이 장에서는 코드 생성 시스템 및 확장기능의 개발과 생성된 결과 코드를 소개한다.

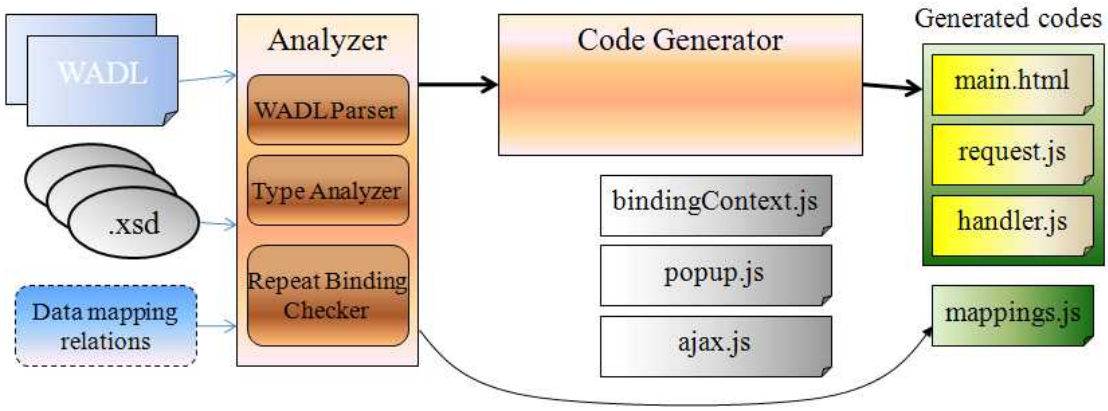
5.1 매시업 코드 자동 생성 시스템

본 논문에서 제안된 방법을 저자들이 개발한 매시업 코드 생성 시스템에 확장한 내용을 소개한다. 저자들은 이전 논문에서 매시업 명세로부터 클라이언트 페이지 코드를 생성하는 MMMash 시스템을 소개하였다[13,14]. 본 논문에서는 제안된 바인딩 컨텍스트를 이용한 이동 설계 방법을 MMMash에 확장하였다. (그림 4)는 코드 생성기와 타입 분석기 등의 모듈을 포함하는 시스템 아키텍처를 보여준다.

이 시스템은 REST 서비스의 표준 명세 언어인 WADL(Web Application Description Language)을 통해 서비스 명세를 입력받는다[15]. WADL에서 사용된 스키마 타입이 정의된 XSD 파일도 같이 읽어들인다. 한편 이 시스템에서는 WADL 파일의 <doc> 요소를 뷰의 타이틀, 항목이나 메뉴 레이블로 사용한다. MMMash 시스템은 HTML과 자바스크립트로 된 결과 코드를 생성한다. 자바스크립트 언어는 웹매시업의 구현 언어로 널리 쓰이고 있으며, 강력한 표현력과 동적인 실행 환경으로 웹 개발 언어로 주목받고 있다[16].

(그림 4)와 같이 코드 생성 모듈은 서비스 명세 WADL 파일과 데이터 매핑 관계를 입력으로 받아들인다. 서비스 명세의 메소드 시그너춰와 XSD 파일의 스키마 정의 타입을 분석하여, 뷰와 통신부 코드를 생성한다. 결과 데이터의 트리 구조에 따라 프리젠테이션 코드 및 요청 및 회신처리부(response handler)의 코드 생성 방법은 기존의 연구에서 잘 알려져 있다. 본 논문에서는 제안된 컨텍스트 메뉴 기반의 이동 설계 기능을 구현하기 위하여 스키마 구조로부터 반복부 트리를 생성하고 바인딩 컨텍스트를 계산하는 부분을 확장하였다.

생성된 코드 결과는 (그림 4)와 같이 HTML 페이지와 여러 개의 JS 파일로 이루어진다. 시스템은 지원되는 모든 서비스 메소드에 대해 요청과 회신처리부 함수를 생성한다. Analyzer 모듈에서는 바인딩 컨텍스트 계산 결과를 "mappings.js" 파일에 자바스크립트 연관 배열 형태로 직렬화한다.



(그림 4) MMMash 시스템의 입력 파일과 코드 생성 결과와 구조

〈표 1〉 바인딩 컨텍스트 예 (m_1=ReadEvent)

m_1	π_r	m_2	p_i	π_i
ReadEvent	/event	SearchMemos	eventId	eventId
		SearchExhibits	eventId	eventId
	/event/part	ReadMap	name	location
		SearchEvents	Location	location
	/event/part/when	CreateReserve	eventId	eventId
			Location	location
			Time	time

5.2 바인딩 컨텍스트를 이용한 동적 메뉴 생성

4장에서 정의된 바인딩 컨텍스트는 결과 뷰의 컨텍스트에서 어떤 메소드의 요청 메뉴가 생성되어야 할지, 그리고 어느 데이터를 입력 매개변수로 보내야 할지를 포함한다. MMMash 시스템의 분석부에서 생성된 바인딩 컨텍스트 정보는 <표 1>와 같은 연관배열 형태로 직렬화된다. 여기서 π_r은 반복부 노드의 경로를 나타낸다.

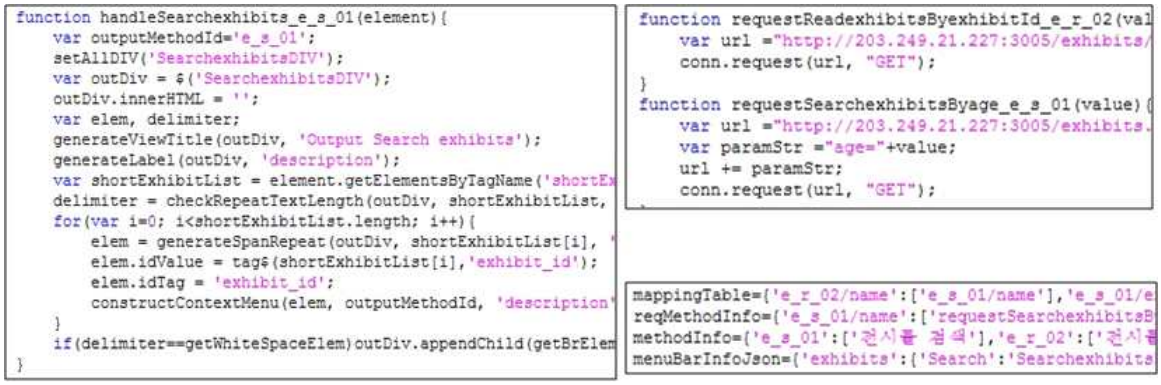
본 시스템에서는 동적으로 생성된 회신처리기 함수에서 컨텍스트 메뉴 생성부를 호출한다. 이 때 바인딩 컨텍스트 정보를 바탕으로 생성할 컨텍스트 메뉴의 종류에 따라 뷰 메뉴 또는 반복부에 대한 컨텍스트 팝업 메뉴를 (그림 2)와 같이 생성하게 된다.

바인딩 컨텍스트로부터 동적으로 메뉴를 생성하는 회신처리기 코드의 알고리즘을 (그림 5)에서 보여준다. (그림 5)의 알고리즘은 트리의 반복부 구조를 중심으로 프리젠테이션 코드를 생성하는 [14]의 방법을 바탕으로 자동생성되는 핸들러 코드 부분이다. 반복부 구조를 이용하여 결과뷰를 생성하기 위하여 단순 데이터 반복인 경우 차례로 나열, 1단계 반복부 트리인 경우 표의 행, 2단계 반복부 트리는 표안의 표, 그 이상은 표를 포함하는 문단 요소의 반복으로 처리한

Algorithm Generating output view and context popup menus

```
1   function handleReadEvent_a_r_02(outputTree)
2       var outputMethodId='a_r_02';
3       var outDiv = $('ReadEventDIV');
4       …;
5       generateViewTitle(outDiv, 'Output Read    Event');
6       generateViewMenus(outdiv, outputTree);
7       for each node n ∈ outputTree의 root의 직접자손터미널
8           elem = generate <span> for tag(n);
9       …
10      create <table> element and table headers;
11      for (rptNode0 in getElements(outputTree, '/event/part'))
12          generate <tr> element;
13          popupMenu0 = constructRptBindingMenu(outputMethodId, rptNode0));
14          elem = generate <td> for getElement(rptNode, 'location');
15          elem.appendEvent('onmouseover', popupContextMenu(popupMenu0));
16          generate <td>;
17          for (rptNode1 in getElements(rptNode, 'time')
18              popupMenu1 = constructRptBindingMenu(outputMethodId, rptNode1);
19              elem = Generate <span> for rptNode1;
20              elem.appendEvent('onmouseover', popupContextMenu(popupMenu1));
21  function constructRptBindingMenu(m1,R)
22      for all m2 s.t. (R,[p1, p2, …, pk]) in bindingContext(m1, m2)
23          menuSpan    = createContextMenuSpan(m2,R);
24          menuSpan.onclick(…);    // request of m2 with values pi(R)
```

(그림 5) 자동생성된 자바스크립트 핸들러 함수의 슈도 코드 (ReadEvent 사례)



(그림 6) 자동생성 코드 부분 (좌측:핸들러함수, 우상단:요청함수, 좌하단:자료구조)

다. 이러한 구조에 컨텍스트 메뉴의 생성을 위한 코드가 추가된다.

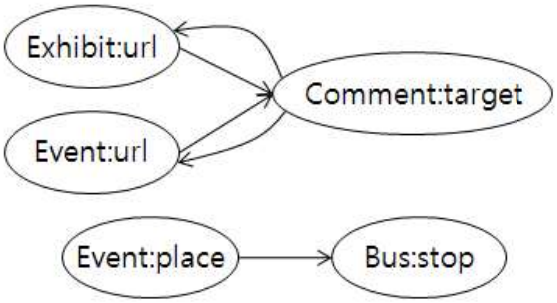
시스템에 의해 자동생성된 ReadEvent 메소드(아이디가 a_r_02임)의 회신 처리기 함수의 슈도 코드이다. 먼저 전체 트리에 대해 루트 노드를 바인딩 컨텍스트에 포함하는 메소드의 호출 메뉴가 뷰 메뉴로 추가된다 (6번줄). 다음으로 <table> 구조가 생성되고, 1단계 반복되는 part 요소에 대해 <tr> 구조를 생성하고 2단계 반복되는 time 요소에 대해 요소를 생성한다. 이 때 14, 17번 줄에서 각 반복부 노드에 대한 컨텍스트 기반 팝업메뉴가 생성된다. 이와 같이 해당 반복부에 대해 생성된 팝업메뉴는 터미널 노드의 텍스트 값에 대한 요소가 생성될 때(14번줄, 19번줄), onmouseover 이벤트로 연결된다.

한편 각 반복부 노드에 대해 바인딩 컨텍스트를 이용하여 팝업 메뉴를 생성하는 부분은 constructRptBindingMenu() 함수에 해당하는데, 이것은 미리 생성된 파일의 정적 코드로 회신처리부 함수에서 호출하게 된다.

5.3 생성 결과 및 분석

본 논문에서 생성한 결과 페이지는 박물관, 관광커뮤니티, 교통 3개의 서비스로 이루어지며 35개의 메소드를 대상으로 하였다. 서비스 내부의 데이터 매핑은 동일한 네임스페이스이므로 태그명 기반으로 가능하며, 서비스 간의 데이터 매핑은 (그림 7(b))와 같이 주어졌다. 이러한 서비스 환경에

(a) 서비스 명세에 정의된 리소스와 메소드 / 생성뷰 개수



(b) 사용자 정의 데이터 매핑 관계

뷰별 메뉴 개수	해당 뷰의 수
2	4
3	5
4	6
5	
6	1

(c) 뷰별 컨텍스트 메뉴의 개수

(그림 7) 사례 시스템의 서비스 환경과 데이터 매핑 및 컨텍스트 메뉴 생성결과 분석

대해 MMMash 시스템에서 생성되는 뷰는 입력부 7개, 결과 뷰 16개이며, 모든 뷰에서 요청 가능한 정적 메뉴에 의한 서비스 요청은 6개이며, 각 뷰 별로 평균 컨텍스트 메뉴의 개수에 때라 분류한 표가 (그림 7(c))와 같다. 여기서 보는 바와 같이 각 뷰 별로 4-5개 정도의 컨텍스트 메뉴가 생성됨을 알 수 있다.

동일한 서비스 내부에서의 매시업에 의한 컨텍스트 메뉴는 [13]에서 제시한 RESTful 서비스의 이동 메뉴 패턴에 따라 예측 가능한 형태로 생성됨을 알 수 있었다. 반면 서비스 간의 매시업 메뉴는 사용자 정의 데이터 매핑에 의해 상당 부분 결정된다. 그러므로 사용자 정의 데이터 매핑을 적절히 정의하여 서비스 간 매시업을 설계할 수 있음을 확인할 수 있었다.

생성된 페이지에서는 컨텍스트에 따라 매시업 가능한 연결 메뉴를 보여주어 메소드가 많은 경우에도 복잡한 매시업 관계를 사용자가 사용하기 쉬운 형태로 표현해 줄 수 있다. 더구나 (그림 2)와 같은 컨텍스트 팝업 메뉴는 사용자가 입력 매개변수 데이터를 선택하거나 입력하는 수고를 덜어주어 편리하다.

6. 결 론

본 논문에서는 동적인 웹서비스 환경과 클라이언트 측 서비스 조합을 지원하는 클라이언트 매시업 페이지의 사용자 인터페이스 개발 방법을 살펴보았다. 특히 매시업 서비스 호출과 이동을 위한 컨텍스트 메뉴의 생성을 위해 메소드 간의 데이터 흐름을 결정하는 방법으로 결정적 바인딩 관계를 정의하였고, 반복부 데이터에 대한 바인딩 컨텍스트의 계산 방법을 소개하였다. 또한 바인딩 컨텍스트를 이용하여 결과 뷰의 현재 컨텍스트에 대해 팝업 메뉴를 생성하는 방법을 제시하였다.

기존의 클라이언트 측 서비스 통합 프레임워크와 비교하면, 제안된 방법은 컨텍스트 메뉴를 통한 서비스 매시업의 사용자 인터페이스를 중심으로 실용적인 모델을 제안하였고, 생성된 결과 페이지 코드가 사용자에게 서비스 조합과 매시업을 편리하게 제어할 수 있는 기능을 제공한다. 결정적 바인딩 관계를 이용한 컨텍스트 메뉴는 메소드 수의 증가에 따라 복잡도가 급증하는 메소드 간의 매시업 관계를 효과적으로 분류하여 편리한 사용자 인터페이스를 생성할 수 있으며, 사용자가 추가의 상호작용 없이 다음 요청으로 데이터를 전달할 수 있다는 장점을 가진다. 본 논문에서는 역사 관광 서비스를 생성하여 제안된 방법의 타당성을 보였다.

본 논문은 몇 가지 제약을 가진다. 우선 생성된 뷰 간의 관계에서 이동 관계만을 고려하고 배치나 뷰의 통합 등은 다루지 않는다. 또한 제안된 방법은 데이터 타입 정의와 매핑을 바탕으로 하는데, 실제 웹서비스 환경에서는 서비스 명세에서 메소드 원형의 타입 정의가 주어지지 않는 경우가 많이 있다. 또한 생략 가능한 매개변수나 클라이언트 시스템에서 내장된 데이터(전화번호, 위치, 비밀번호 등)를 활용하는 것도 실제 매시업의 개발에서 해결되어야 하는 중요한 문제이다. 이외에도 서비스 검색과 조합 단계가 자동화되어야 동적인 서비스 환경에 적합한 클라이언트의 생성이 가능할 것이다. 이러한 점을 극복하기 위해 본 연구진에서는 MMMash 시스템을 실제 환경의 다양한 문제를 다룰 수 있도록 개발 플랫폼의 형태로 발전시킬 계획이다.

References

[1] Yu, J. et al.: Understanding mashup development. IEEE Internet computing. Vol.12 No.5. pp.44-52, 2008.

[2] Stefan Pietschmann, Tobias Nestler, Florian Daniel, “Application composition at the presentation layer: alternatives and open issues,” iiWAS’2010.

[3] Stefan Pietschmann, Martin Voigt, Klaus Meibner, “Dynamic composition of service-oriented web user interfaces,” ICIW’2009, pp.24-28, 2009.

[4] Florian Daniel, Fabio Casati, Boualem Benatallah, Ming-Chien Shan, “Hosted universal composition: models, languages and infrastructure in MashArt,” ER’2009, pp.428-443, 2009.

[5] Stefan Pietschmann, Johannes Waltsgott, Klaus Meißner, “A Thin-Server Runtime Platform for Composite Web Applications,” ICIW’10, pp.390-395, 2010.

[6] Lecue Freddy Lecue, Samir Salibi, Philippe Bron, Aur?lien Moreau, “Semantic and Syntactic Data Flow in Web Service Composition,” Proc. of IEEE International Conference on Web Services, pp.211-218, 2008.

[7] G. Castelli, M. Mamei, F. Zambonelli, "The changing role of pervasive middleware: from discovery and orchestration to recommendation and planning," 8-th Percom Workshop on middleware and system support for pervasive computing, 2011.

[8] G.Li, H.Sun, "RESTful dynamic framework for services in mobile wireless networks," EBISS'2009, Wuhan, China, 2009.

[9] Jhingran, A.: Enterprise information mashups:integrating information, simply. VLDB’06, pp.3-4, 2006.

[10] Freddy Lecue, Eduardo Silva, Luis Ferreira Pires, “A framework for dynamic web services composition,” Wewst’2007. 59-75.

[11] Ohad Greensphan, Tova Milo, Neoklis Polyztis, “Auto completion for mashups (MatchUp),” Proc. of VLDB’2009, pp.538-549, 2009.

[12] Tobias Nestler, “Towards a Mashup-driven End-User Programming of SOA-based Applications,” iiWAS ’08, pp.551-554, 2008.

[13] Arto Salminen, Feetu Nyrhinen, Antero Taivalsaari, “Developing client-side mashups: experiences, guidelines and the road ahead,” MindTrek’2010. (Filand)

[14] ---, “REST 서비스 패턴을 이용한 매시업 클라이언트 뷰 이동 코드 생성,” 정보처리학회 논문지D, 17-D권, 6호, 2010.

[15] ---, “서비스 조합을 위한 XForms 기반의 모바일 사용자 인터페이스 개발,” 정보처리학회 논문지D, 15-D권, 6호, pp.879-888, 2008.
[16] Danial Deutch, Ohad Greensphan, Tova Milo, “Navigating in complex mashed-up applications,” VLDB’2010.(2010) 320-329
[17] Web Application Description Language(WADL), http://www.w3.org/Submission/wadl.
[18] W3C, "Scripting and AJAX," http://www.w3.org/standards/webdesign/script.

이 은 정
e-mail : ejlee@kyonggi.ac.kr
1988년 서울대학교 계산통계학과(학사)
1990년 한국과학기술원 전자계산학과 (공학석사)
1994년 한국과학기술원 전자계산학과 (공학박사)
1994년~2000년 전자통신연구원 선임연구원
2001년~현 재 경기대학교 컴퓨터과학과 부교수
관심분야 : XML 처리 기술, 웹서비스