클라이언트-서버 통합 웹 프로그래밍언어

전남대학교 ▌최광훈

1. 서 론

다계층 통합 웹 프로그래밍언어(Multi-tier web programming languages)는 클라이언트, 서버, 데이터베이스 계층을 하나의 프로그램으로 개발할 수 있는 방법을 제공한다. 웹 시스템은 데이터베이스와 연동하는 웹 서버와 사용자 인터페이스를 제공하는 웹 클라이언트로 구성되어 있고, 네트워크를 통해 서로 연결한다. 서버 컴퓨터와 클라이언트 브라우저에서 동작하는 두 프로그램을 따로 만들어야 하므로 하나의 프로그램을 개발할 때 보다 더 많은 비용이 들고, 두 프로그램을 함께 테스트해야하기 때문에 컴퓨터 한 대에서 실행해 프로그램을 테스트할 때보다 더 복잡해지며, 어느 프로그램 하나가 변경되면 다른 프로그램과의 정합성도 유지 보수해야 하는 번거로움이 있다. 서버와 클라이언트 컴퓨터의 경계로 나누기 어려운 작업을 개발하면 관련 클라이언트 모듈과 서버 모듈 간의 결합도(coupling)가 높아지는 문제점도 발생한다.

다계층 통합 프로그래밍에서 이러한 문제들을 해결한다. 개발자가 클라이언트 서버 프로그램을 단일 프로그래밍언어로 하나의 통합 프로그램으로 작성하면 컴파일러가 계층별 프로그램으로 자동 분리하고 계층 간 함수 호출을 컴파일러가 추가한 원격 함수 호출(RPC, remote procedure call)로 연결하는 코드를 생성한다.

이 글에서는 클라이언트 서버 모델에서 동작하는 웹 프로그램 개발을 목적으로 설계된 클라이언트 서버 통합 프로그래밍언어에 관한 연구 동향을 소개한다. 통합 프로그래밍언어로 Links [2], Hop [4], Eliom [5,6], Ur/Web [1], ScalaLoci [7], Gavial [11], Opa [10]가 있고, 통합 프로그래밍언어 원리에 관한 연구로 ML5 [3], 다계층 계산법 (Multi-tier calculus [13,14])과 원격함수호출 계산법 (Rpc calculus [12,15,16])과 같은 연구가 진행되었다.

클라이언트 서버 통합 웹 프로그래밍언어의 공통 특징으로 통합 계층 및 슬라이싱 컴파일, 내재된 HTML, 데이터베이스, 반응성 (reactivity)이 있다.

제2절에서는 기존의 클라이언트-서버 통합 웹 프로그래밍언어들의 네 가지 특징을 소개하고, 제3절에서는 저자가 연구했던 원격함수호출 계산법에 대해 소개하고, 제4절에서 향후연구 주제를 나열한다.

2. 클라이언트-서버 통합 웹 프로그래밍언어

Links [2]로 작성한 예제 프로그램을 통해서 클라이언트-서버 통합 웹 프로그램의 특징들을 간단히 살펴본다.

그림 1의 프로그레스바 예제 프로그램은 클라이언트 함수와 서버 함수를 하나의 프로그램으로 작성할 수 있는 통합 계층 프로그래밍의 특징을 보여준다. 함수정의 인자선언 다음에 client와 server 키워드로 해당 함수를 실행할 위치를 각각 클라이언트와 서버로 지정한다. 예를 들어, Line 1의 showProgress는 클라이언트 함수, Line 7의 compute는 서버 함수, Line 14의 showAnswer와 Line 18의 main은 둘 다 클라이언트 함수이다.

함수 main을 실행하면 초기 웹 화면에 텍스트 입력창과 제출 버튼을 포함한 폼(form)을 표시한다. 숫자를 입력하고 버튼을 누르면 폼의 onsubmit 속성으로 지정한 (중괄호로 감싼) 코드가 실행된다. 즉, 서버 함수 compute를 호출하여 파일 다운로드와 같은 시간이 오래 걸리는 어떤 계산을 수행한 다음 클라이언트 함수 showAnswer를 통해 웹 화면에 결과를 표시한다. 서버 함수 compute를 실행하는 중에 사용자에게 진행사항을 알리기 위해서 클라이언트 함수 showProgress를 단위 시간마다 호출하여 프로그레스바를 점진적으로 그린다.

이와 같이 클라이언트와 서버의 코드를 단일 프로그래밍언어로 프로그램 하나에서 함께 작성할 수 있다. 컴파일러는 이 프로그램을 클라이언트와 서버 위치에 각각

† 이 논문은 2019년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2019R1I1A3A01058608)

```
01: fun showProgress(count, total) client {
02:   var percent = 100.0 *. intToFloat(count) /. intToFloat(total);
03:   replaceNode( <div id="bar" style="width:{floatToString(percent)}%;
04:                                 background-color: black">|</div>, getNodeById("bar") )
05: }
06:
07: fun compute(count, total) server {
08:   if (count < total) {
09:     showProgress(count, total);
10:     compute(count+1, total)
11:   } else "done counting to " ^^ intToString(total)
12: }
13:
14: fun showAnswer(answer) client {
15:   replaceNode( <div id="bar">{stringToXml(answer)}</div>, getNodeById("bar") )
16: }
17:
18: fun main() client {
19:     page
20:       <html>
21:         <body>
22:           <form l:onsubmit="{ showAnswer(compute(0, stringToInt(n))) }">
23:             <input type="text" l:name="n"/>
24:             <input type="submit"/>
25:           </form>
26:           <div id="bar"/>
27:         </body>
28:       </html>
29: }
30:
31: main()
```

그림 1 Links 예제 프로그램 (Progress Bar)

배치할 코드를 분리하고, 함수호출자(caller)와 피호출함수(callee)가 서로 다른 곳에 배치된 경우 네트워크 통신 라이브러리를 사용하는 원격함수 호출로 대체한다. 통합 프로그램을 위치에 따른 부분 프로그램으로 분리하는 과정을 슬라이싱 (slicing) 컴파일이라 부른다.

Links 프로그래밍언어는 클라이언트-서버 모델을 위한 원격함수 계산법(Remote procedure call calculus) [12]을 기반으로 위치지정 특징을 설계하였다. 이 계산법은 위치 정보를 타입으로 표현한 타입기반 원격함수 계산법(Typed RPC calculus) [15]와 위치에 무관한 코드의 경우 클라이언트와 서버 위치를 모두 지정할 수 있는 위치 지정 다형성을 지원하는 다형타입 원격함수 계산법(Polymorphically typed RPC calculus) [16]으로 발전되었다.

모든 클라이언트 서버 통합 프로그래밍언어가 이러한 특징을 가지고 있는 것은 아니다. Links는 원격함수 계산법에 따라 클라이언트와 서버가 서로의 함수를 자유롭게 호출할 수 있도록 설계되었다. ML5는 모달 로직(modal logic)의 원리를 기반으로 설계되었고, 클라이언트와 서버가 서로 대칭적으로 자유롭게 호출할 수 있다. 하지만 Hop, Eliom, Ur/Web은 클라이언트에서 서버를 호출하는 방향만 가능하다.

클라이언트-서버 통합 웹 프로그래밍언어는 일반적으로 웹 프로그램의 HTML을 다루는 방법을 라이브러리가 아닌 언어의 특징으로 제공한다. Links에서 HTML을 마치 정수와 문자열과 같은 기본 자료형처럼 다룰 수 있는 구문과 타입을 제공한다. [그림 1]의 함수 main에서 태그 html, body, form, input, div를 그대로 사용하여 웹 페이지에 표시될 HTML 문서를 만든다. 중괄호로 감싼 코드를 담은 문자열로 폼에서 실행할 코드를 HTML 문서에 끼워 넣을 수 있다.

Links는 HTML 폼을 추상화한 특징 폼릿(formlet)을 제공한다[17]. 폼릿을 통해 기존 HTML을 언어에 내재시킬 때 발생하였던 한계를 극복하였다. 첫째, HTML의 입력 폼과 입력 후 처리 코드의 연결을 명시적으로 표현하고, 둘째, 둘 이상의 입력 폼들을 조합하여 구조화된 데이터 입력을 처리할 수 있고, 셋째, 동일한 입력 폼을 라이브러리 함수처럼 한 번 정의하고 동일한 문서에서 두 번 이상 사용할 수 있다.

Hop, Eliom, Ur/Web에서도 비슷하게 언어에 내재된 HTML 특징을 제공한다. 특히 Ur/Web은 의존타입(dependent type)을 사용하는 라이브러리를 제공하고 개발자가 이 라이브러리를 사용하여 원하는 HTML을 모델링한다. Ur/Web 컴파일러에서 HTML 생성 및 조작을 자동 타입 검사로 검증하여 문자열을 통한 코드 주입 취약점(code injection vulnerability)과 같은 보안 문제를 사전에 방지할 수 있다[18].

웹 프로그래밍에서 데이터베이스 사용은 필수적이다. 통합 웹 프로그래밍언어는 데이터베이스 계층을 지원하는 특징을 제공한다. Links는 Kleisli가 제안한 기법[19]에 기반한 데이터베이스 질의(Query) 방법을 제공한다. 다음 예제 프로그램을 살펴보자[20].

[그림 2]의 함수 filterEmployees는 조건을 테스트하는 함수 p를 받아 데이터베이스 테이블 employees에서 이 조건을 만족하는 레코드를 선택한 다음 선택된 레코들들에서 이름만 뽑아 리턴한다. 이 함수에 월급이 20000보다 적은지 테스트하는 (이름 없는) 람다 함수를 적용하였다.

이 데이터베이스 프로그래밍 예제에서 보여주듯이 SQL과 함수를 자유롭게 조합하여 프로그램을 작성할 수 있다. 다만 컴파일러는 SQL에서 허용되지 않는 함수를 사용할 경우 자동으로 검사하여 컴파일 에러를 낸다. 예를 들어 [그림 2] 예제에서 사용한 고용인의 원급과 상수 20000을 비교하는 함수를 SQL에서 지원할 수 있기 때문에 문제가 발생하지 않는다. [그림 2] 예제에서 Line 6을 아래의 코드로 바꿀 경우 Links 컴파일러는 컴파일 에러를 낸다.

```
filterEmployees (lambda ( e ). (primes(e.salary) < 20000))
```

여기서 primes(n)은 주어진 인자 n보다 작은 소수의 개수를 반환하는 함수이다. SQL 기본함수(primitives)로 primes 함수를 직접 지원하지 않는다. 따라서 SQL 문 실행 중간에 사용자정의 함수를 호출하고 그 반환 결과를 받아 그 실행을 이어가야 하는데 데이터베이스 시스템에서 이러한 유형의 실행을 지원하지 않는다.

Links의 타입시스템은 SQL문을 사용하는 코드와 일반 프로그램 코드를 구분하는 열 기반 이펙트(Row-based effects) 타입 시스템을 이용하여 데이터베이스 계층과 서버 계층을 하나의 프로그램으로 작성하는 것을 허용하면서도 위의 예제에서 설명한 것과 같은 오류를 자동으로 찾아낸다.

Ur/Web은 HTML을 지원하는 것과 동일한 방식으로 언어에 내재된 SQL문을 지원한다. 앞서 HTML에 대해서 설명했던바와 같이 의존타입을 갖는 라이브러리를 이용하여 개발자는 SQL문을 작성하고 Ur/Web 컴파일러는 타입 검사를 통해 작성된 SQL문의 유효성을 자동으로 확인한다. 따라서 SQL문 주입 취약점(SQL injection vulnerability)을 사전에 방지할 수 있다.

이 외의 클라이언트 서버 통합 웹 프로그래밍언어 Hop과 Eliom은 문자열로 SQL문을 작성하거나 준비된 라이브러리를 통해서 SQL 질의를 하도록 설계되었다.

일반적으로 웹 프로그램은 사용자 입력과 클라이언트 서버 통신은 비동기 방식을 따르고, 클라이언트, 서버, 원격 서비스가 동시에 여러 작업을 실행하는 시나리오가 빈번하다. 이러한 방식을 자연스럽게 프로그래밍 할 수

```
01: filterEmployees ( p ) =
02:     query (for e <- asList employees)
03:               where ( p(e) )
04:                 [<name = e.name>]
05:
06: filterEmployees (lambda ( e ). (e.salary < 20000))
```

그림 2 Links 데이터베이스 프로그래밍 예제

있도록 반응성(reactivity) 언어 특징이 제안되었다.

반응성 언어 특징은 외부 이벤트에 대한 반응을 기본 개념으로 프로그램을 구조화하는 프로그래밍 방식이다. 마우스, 키보드, 터치 이벤트를 각각 이벤트 핸들러로 처리하는 방식을 사용할 때 이벤트 핸들러 수가 많아지면 실행 흐름이 프로그램 구조에서 드러나지 않아 제대로 작성하거나 이해하기 어렵다. 반응성 특징으로 이벤트 핸들러의 문제점을 해결할 수 있다. 반응성 특징을 지원하는 대표적인 방법으로 FRP (Functional reactive programming) [24]와 Arrow [25][26]가 있다. 더욱 직관적인 형태의 반응성 특징으로 클라이언트 웹 프로그래밍언어 Elm에서 제공하는 Model-View-Update가 있다[21].

다음은 Model-View-Update 구조를 Links로 작성한 예제 프로그램이다[22].

[그림 3]의 예제 프로그램은 사용자가 입력 폼에 텍스트를 입력할 때마다 뒤집힌 텍스트 내용을 표시한다. Line 16의 메인 페이지 함수에서 모델, 뷰 함수, 업데이트 함수를 Line 19에서 지정한다.

Line 1에서 모델의 레코드타입을 선언한다. Links에서 소괄호는 레코드 타입 또는 레코드 값이다. 모델은 키contents와 문자열 값을 갖는 레코드이다. Line 19에서 (contents="")와 같이 이 레코드의 문자열을 공백으로 지정한 모델로 초기화한다.

뷰와 업데이트 함수는 각각 view와 updt이다. Line 2에서 뷰 함수와 업데이트 함수가 사용할 메시지 타입을 선언한다. Links에서[|과 |]는 데이터타입 선언 구문이다. UpdateBox는 메시지를 구분하는 태그이고, 이 태그의 메시지에 문자열이 있음을 선언한다. Line 04의 뷰 함수는 모델을 인자로 받고 HTML (몸체 내용)과 메시지(폼 입력 값)을 반환하고, Line 11의 업데이트 함수는 메시지와 현재 모델을 받아 메시지 내용에 따라 새로운 모델을 만들어 반환한다. 뷰 함수에서 만들어 반환하는 HTML 내용은 입력 폼과 현재 모델에 저장된 문자열을 뒤집어서 만든 텍스트 노드이다.

예제에서 소개한 Links의 특징은 클라이언트 안에서만 동작하는 반응성이다. 이와 달리 ScalaLoci와 Gavial은 클라이언트와 서버가 모두 연동하는 다계층 반응성(Multi-tier reactivity) 특징을 제공한다[7][11]. 채팅 웹 프로그램을 작성할 때 상대방이 작성한 메시지는 일단 서버로 전달되고 이 서버는 나머지 모든 클라이언트들에 전달한다. 이때 클라이언트와 서버 사이에 양방향으로

```
01: typename Model = (contents: String)
02: typename Message = [| UpdateBox: String |]
03: sig view : (Model) ~> HTML(Message)
04: fun view(model) {
05:   vdom
06:     <input type="text" value="{model.contents}"
07:       e:onInput="{fun(str) { UpdateBox(str) }}" />
08:     <div> { textNode(reverse(model.contens)) } </div>
09: }
10:
11: sig updt : (Message, Model) ~> Model
12: fun updt(UpdateBox(newStr), model) {
13:   (contents = newStr)
14: }
15:
16: fun mainPage() {
17:   page
18:     <html><body>
19:       {| mvuPage( (contents=""), view, updt) |}
20:     </body></html>
21: }
```

그림 3 Links의 모델−뷰−업데이트 구조의 리액티브 프로그램 예제

이벤트를 전달하고 처리한다. 이벤트 처리 방식은 클라이언트 안에서만 동작할 때와 다르지 않지만 서로 다른 컴퓨터에서 이벤트 전달 시간이 달라 프로그램 상으로 동일한 값이어야 하는데 실행할 때 서로 다른 값이 되는 불일치 문제(glitch)가 발생할 수 있다.

Gavial은 스칼라 기반으로 작성한 다계층 리액티브 프레임워크로, 클라이언트와 서버 안에서 지역 불일치 해소 (local glitch freedom)와 동일한 클록을 공유한 전역 불일치 해소 (global glitch freedom)를 타협한 계층 불일치 해소 (tiered glitch freedom)를 제공한다. Gavial 프로그래밍 모델은 클라이언트와 서버간의 요청-응답 스타일, CRUD 스타일, 양방향 상호작용에 적합하다. 클라이언트와 클라이언트가 직접 통신하는 형태는 지원하지 않는다.

최근 Links는 세션 타입(session type)에 기초한 계산법을 통하여 반응성과 동시성을 지원하게 되었다 [22, 23]. 세션 타입 계산법은 새로운 스레드를 만들고 이 스레드와 새로운 세션을 만들어 정해진 프로토콜로 통신한 다음 세션을 닫고 스레드를 종료하는 유형으로 작성한 프로그램을 자동으로 검증할 수 있다. 이 세션 계산법으로 앞서 소개했던 [그림 3]의 모델-뷰-업데이트 구조의 반응성 프로그래밍을 지원할 뿐만 아니라 임의로 여러 스레드를 만들고 메시지 기반으로 통신하는 프로그램을 작성할 수 있다. 서버와 클라이언트뿐만 아니라 클라이언트와 클라이언트 사이에도 자유롭게 메시지를 주고받을 수 있다.

3. 원격함수호출 계산법 (RPC calculus)

이 절에서 클라이언트-서버 통합 웹 프로그램의 특징 중 하나인 계층 간 함수 호출의 의미(semantics)와 컴파일 방법에 관하여 자세히 살펴본다. Cooper와 Wadler는 원격함수호출 계산법 (RPC calculus)를 제안하여 계층 간 함수 호출을 자유롭게 표현하는 람다 계산법을 정의하였다[24].

[그림 4]는 클라이언트 서버 모델을 위한 원격함수호출 계산법의 구문과 의미를 정의한다. 람다 계산법과 동일한 구문을 사용하되 람다식(lambda expression)에 클라이언트 (c) 또는 서버(s)를 붙여 해당 람다식 $\lambda^a x \cdot N$(의 몸체)를 실행할 위치 a를 지정한다.

원격함수호출 계산법의 실행 의미는 빅스텝 오퍼레이셔널 시맨틱스(big-step operational semantics) 스타일로 정의되었다. 일반 람다계산법의 실행 의미와 매우 흡사하지만, 람다식 M을 실행하여 V를 내는 실행 판단(evaluation judgment), $M \Downarrow_a V$에 현재 실행 위치 a를 지정하였다. 의미 규칙 (Beta)는 람다적용식(lambda application) L M을 위치 a에서 실행하여 그 결과로 V를 내는 과정을 정의한다. L과 M을 동일한 위치에서 각각 계산하여 람다식과 어떤 값 W를 얻는다. 이 람다식에 위치 b가 지정되어 있으므로 위치 b에서 x를 W로 대체한 람다식의 몸체를 계산한다.

이때 람다적용식 L M의 함수호출자 위치는 a이고, 피호출함수 위치는 b이다. 위치 a와 위치 b가 동일하면 동일한 위치(계층)에 있는 (지역)함수를 호출하고, a와 b가 다르면 다른 위치(계층)에 있는 원격함수를 호출한다.

원격함수호출 계산법의 구문으로 클라이언트-서버 통합 프로그램 예제를 아래와 같이 작성할 수 있다.

$$\left(\lambda^{\mathbf{s}} f \cdot \left(\lambda^{\mathbf{s}} x \cdot x\right)\left(f\ c\right)\right)\ \left(\lambda^{\mathbf{c}} y \cdot \left(\lambda^{\mathbf{s}} z \cdot z\right)\ y\right)$$

이 예제를 의미 규칙에 따라 실행하면 람다적용식 (f c)는 서버에서 클라이언트를 호출하는 원격함수호출이다. 그 이유는 이 식은 서버로 지정한 람다식의 몸체에 있기 때문에 함수호출자는 서버이고, 피호출함수 f는 클라이언트로 지정한 람다식이기 때문이다.

원격함수호출 계산법은 지역함수호출과 원격함수호출을 구문에서 구분하지 않고 모두 람다적용식(L M)으로 표현하는 특징이 있다. 원격함수호출을 구분하는 별도의 키워드 (예: rpc L M)를 사용하지 않아 계산법이 간결한 장점이 있다. 하지만 L을 실행하기

Syntax

Location	a,b	::=	$\mathbf{c} \mid \mathbf{s}$
Term	L,M,N	::=	$V \mid L\,M$
Value	V,W	::=	$x \mid \lambda^a x \cdot N$

Semantics

$$\frac{}{V \Downarrow_a V}\ (\text{Value}) \qquad \frac{L \Downarrow_a \lambda^b x \cdot N \qquad M \Downarrow_a W \qquad N\{W/x\} \Downarrow_b V}{L\,M \Downarrow_a V}\ (\text{Beta})$$

그림 4 원격함수호출 계산법 (RPC calculus)의 구문과 의미

Types

Type τ ::= *base* | $\tau \xrightarrow{a} \tau$

Typing Rules

$$\text{(T-Var)}\ \frac{\Gamma(x) = \tau}{\Gamma \rhd_a x : \tau} \qquad \text{(T-Lam)}\ \frac{\Gamma\{x:\tau\} \rhd_b M : \tau'}{\Gamma \rhd_a \lambda^b x.M : \tau \xrightarrow{b} \tau'} \qquad \text{(T-App)}\ \frac{\Gamma \rhd_a L : \tau \xrightarrow{a} \tau' \quad \Gamma \rhd_a M : \tau}{\Gamma \rhd_a L\,M : \tau'}$$

$$\text{(T-Req)}\ \frac{\Gamma \rhd_{\mathbf{c}} L : \tau \xrightarrow{\mathbf{s}} \tau' \quad \Gamma \rhd_{\mathbf{c}} M : \tau}{\Gamma \rhd_{\mathbf{c}} L\,M : \tau'} \qquad \text{(T-Call)}\ \frac{\Gamma \rhd_{\mathbf{s}} L : \tau \xrightarrow{\mathbf{c}} \tau' \quad \Gamma \rhd_{\mathbf{s}} M : \tau}{\Gamma \rhd_{\mathbf{s}} L\,M : \tau'}$$

그림 5 타입기반 원격함수호출 계산법(typed RPC calculus)의 타입과 타입규칙

$$\begin{aligned}
\phi_{\mathbf{c}} \;:\; & main = \mathsf{let}\ r_3 = \mathsf{req}(clo(g_3,\{\}),\ clo(g_5,\{\}))\ \mathsf{in}\ r_3 \\
& g_2 = \{f_7\}\ \lambda^{\mathbf{c}} z_{10}.\ \mathsf{let}\ y_9 = f_7\ z_{10}\ \mathsf{in}\ \mathsf{ret}(y_9) \\
& g_5 = \{\}\ \lambda^{\mathbf{c}} y.\ \mathsf{let}\ r_{14} = \mathsf{req}(clo(g_4,\{\}),\ y)\ \mathsf{in}\ r_{14} \\
\phi_{\mathbf{s}} \;:\; & g_1 = \{\}\ \lambda^{\mathbf{s}} x.\ x \\
& g_3 = \{\}\ \lambda^{\mathbf{s}} f.\ \mathsf{let}\ x_5 = (\mathsf{let}\ r_{11} = \mathsf{call}(clo(g_2,\{f\}),\ c)\ \mathsf{in}\ r_{11})\ \mathsf{in}\ \mathsf{let}\ r_6 = clo(g_1,\{\})\ x_5\ \mathsf{in}\ r_6 \\
& g_4 = \{\}\ \lambda^{\mathbf{s}} z.\ z
\end{aligned}$$

그림 6 슬라이싱 컴파일 결과

전까지 람다적용식이 지역함수호출인지 원격함수호출인지를 구분하지 못하는 단점도 있다. L을 실행한 결과로 받은 람다식의 위치 정보를 얻은 다음에 판단할 수 있다. 비록 동일한 구문으로 두 형태의 함수호출을 표현했지만 실제로 구현할 때는 명확히 구분해야한다. 왜냐하면 지역함수호출은 일반적인 함수호출 방식을 그대로 사용하지만, 원격함수호출은 통신 라이브러리를 사용하여 인자와 호출할 함수 정보를 네트워크를 통해 전달하고 실행한 결과를 다시 받아오는 과정이 필요하기 때문이다.

저자의 연구에서 함수 타입에 위치 정보를 붙여 람다적용식이 원격함수호출인지 타입 검사를 통해 실행하지 않고 분석하는 타입시스템을 제안하였다[15]. 아래의 위치 정보가 있는 함수 타입(located function type)을 갖는 함수(의 몸체)는 반드시 위치 a에서 실행한다.

$$\tau \xrightarrow{a} \tau'$$

[그림 5]는 타입기반 원격함수호출 계산법(typed RPC calculus)의 타입시스템이다.

이 타입시스템에서는 위치 정보가 중요하다. 원격함수호출 계산법에서 람다식에 위치 정보를 붙인 것처럼 함수 타입에 위치 정보를 붙이고, 의미규칙의 실행 판단(evaluation judgment)에 위치 정보를 붙인 것과 같이 타입 판단(typing judgment)에 (삼각형 아래첨자로) 위치 정보를 붙인다.

타입 판단 $\Gamma \rhd_a M : \tau$은 타입환경(type environment) 감마, 위치 a, 람다식 M, 타입 타우의 4가지 원소의 관계(quadruple relation)으로 작성한다. 이때 람다식을 실행하는 위치가 a임을 지정한다.

[그림 5]의 타입규칙을 적용하면 람다적용식에 어떤 타입 규칙을 적용해야하는지에 따라 지역함수호출과 원격함수호출을 구분할 수 있고, 또한 클라이언트에서 서버를 호출하는 경우인지 그 반대인지를 분석할 수 있다. 만일 타입 규칙 (T-App)를 람다적용식의 타입을 구하기 위해 사용했다면 현재 실행하고 있는 위치도 a이고 호출할 함수를 실행할 위치도 a이므로 (함수 타입에 위치 a가 붙어 있으므로) 지역함수호출이다. 타입 규칙 (T-Req)를 사용했다면 함수 타입에 s가 있고 타입 판단에 c가 지정되어 있으므로 클라이언트에서 서버를 호출한다. 타입 규칙 (T-Call)을 사용했다면, 반대로 서버에서 클라이언트를 호출하는 원격함수호출로 분석할 수 있다.

앞서 원격함수호출 계산법의 구문으로 작성한 클라이언트-서버 통합 프로그램 예제에 이 타입규칙을 적용하면 아래와 같은 결과를 얻는다. 람다적용식 (L M) 중간에 위치 정보 a를 붙인 표기법 ($L\ ^a M$)은 이 적용식에서 피호출 함수(L로부터 얻은 람다식)를 실행할 위치가 a임을 나타낸다.

$$\left(\lambda^{\mathbf{s}} f \cdot\ (\lambda^{\mathbf{s}} x \cdot x)\ ^{\mathbf{s}} (f\ ^{\mathbf{c}} c)\right)\ ^{\mathbf{s}} \left(\lambda^{\mathbf{c}} y \cdot\ (\lambda^{\mathbf{s}} z \cdot z)\ ^{\mathbf{s}} y\right)$$

즉, 람다적용식 (f c) 중간에 클라이언트 위치 (c)가 있고, 이 식은 서버 위치에서 실행하고 있으므로 원격함수호출임을 타입 정보로 분석할 수 있다. 따라서 이 프로그램을 실행하기 전에 타입규칙을 적용하면 각

람다적용식의 종류를 분석할 수 있다.

타입기반 원격함수호출 계산법의 핵심 아이디어는 프로그래머는 람다식에 위치 정보를 붙이면 람다적용식에서 호출할 함수의 위치 정보는 타입검사 과정에서 분석할 수 있다는 점이다. 이렇게 분석한 원격함수호출 여부를 이용하여 슬라이싱 컴파일러를 설계할 수 있다.

[그림 6]은 타입 규칙을 적용하여 얻은 위의 예제를 슬라이싱 컴파일한 결과를 보여준다[15].

슬라이싱은 클라이언트-서버 통합 프로그램을 각 위치별 코드로 분리하는 컴파일 과정이다. ϕ_c는 클라이언트 함수들만을 포함하고, ϕ_s는 서버 함수들만을 포함한다. 이 두 위치로 분리된 함수들을 원격함수호출 req와 call로 연결한다. req는 클라이언트에서 서버를 호출하고, call은 그 반대 방향으로 호출한다.

타입기반 원격함수호출 계산법의 제한점은 모든 함수가 클라이언트 또는 서버로 반드시 지정되어야 하는 점이다. 위치에 상관없이 사용할 수 있는 map과 같은 범용함수를 두 위치에 따로 만들어야하는 불편함이 있다.

저자의 연구에서 다형 원격함수호출 계산법(polymorphic RPC calculus)[16]을 제안하여 타입 추상화(type abstraction)와 타입 적용(type application)으로 다형타입(polymorphic type)을 지원하는 것과 같은 방식으로 위치 추상화(location abstraction)와 위치 적용(location application)을 표현할 수 있다. 이 계산법을 사용하면 map과 같은 위치에 무관하게 사용할 수 있는 함수를 위치 다형타입을 갖도록 정의할 수 있다. 위치에 상관없이 하나의 함수를 정의하면 모든 위치에서 사용할 수 있기 때문에 앞서 타입기반 원격함수호출 계산법의 문제점을 해결하였다.

타입기반 원격함수호출 계산법에서 제안한 타입시스템과 슬라이싱 컴파일 방법에 대해서 모두 타입 건전성(type soundness)와 컴파일 방법의 타입 정확성(type correctness)과 의미 정확성(semantics correctness)이 증명되었다. 다형타입 원격함수호출 계산법에서 제안한 의미와 타입시스템간의 타입 건전성도 성립함을 보였고, 다형타입 원격함수호출 계산법으로 작성한 다형위치 통합 프로그램을 타입기반 원격함수호출 계산법으로 작성한 타입기반 통합 프로그램으로 변환하는 다형타입제거 변환(monomorphization)의 타입 정확성과 의미 정확성도 증명되었다[15, 16].

4. 요약 및 향후 연구 주제

Links [2]로 작성한 예제 프로그램을 통해서 클라이언트-서버 통합 웹 프로그램의 4가지 특징들, 통합계층 프로그래밍과 슬라이싱 컴파일, 내재된 HTML, 데이터베이스, 반응성에 대해 간략히 살펴보았다.

향후 연구로 다음과 같은 주제가 있다. 첫째, 통합계층을 지원하는 모듈과 다계층 반응성에 대한 이론적 기반 연구가 필요하다. Eliom과 ScalaLoci은 통합계층 모듈 특징으로 클라이언트 함수와 서버 함수를 함께 모듈화하는 방법을 제안하였으나 모두 이론적 토대가 부족하다. 둘째, Gavial은 다계층 반응성 특징을 구현하고 동작의 불일치 문제 해소와 같은 성질들을 제공한다고 주장하지만 아직 증명이 부족하다. 셋째, 다계층 웹 프로그램이 보안이나 개인정보 보호 문제를 정형화하기에 매우 적합하다. 전통적인 웹 프로그램의 경우 각 계층 별로 프로그램이 따로 있고 작성에 사용한 프로그래밍언어도 다르기 때문에 분석이 어려웠다. 특히 통합 프로그래밍언어에서 표시한 위치정보는 추상적이기 때문에 여러 의미로 해석이 가능하다. 예를 들어 다자간 계산(Multi-Party Computation)에서 위치 정보를 각 참여자로 해석할 수 있다. 다수의 사용자가 각자의 비밀 값을 입력으로 함수를 계산하는 다자간 계산에서 서로의 정보를 누출하지 않는지 보안을 검증하는 문제에서 계층의 위치 정합성을 보장하는 타입 기반 방법을 응용할 수 있다.

참고문헌

[1] A. Chlipala, “Ur/Web: a simple model for programming the web”, Proceedings of the 42nd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL ‘15), vol.50, no.1, pp.153-165, New York, NY, USA, 2015.

[2] E. K. Cooper, S. Lindley, P. Wadler, and J. Yallop, “Links: web programming without tiers”, Proceedings of the 5th Int’l Conf. on Formal Methods for Components and Objects (FMCO ‘06), pp.266-296, Amsterdam, Netherlands, 2006.

[3] T. VII Murphy, “Modal types for mobile code”, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.

[4] M. Serrano and G. Berry. “Multitier programming in hop”, Communication of ACM (CACM), vol. 55, no.8, pp.53-59, August 2012.

[5] G. Radanne, “Tierless web programming in ML”, Ph.D. thesis, Univ. of Paris Diderot, 2017.

[6] G. Radanne and J. Vouillon “Tierless web programming in the large”, Companion Proceedings of the Web Conference, pp.681-689, Lyon, France, 2018.

[7] P. Weisanburger, M. Köhler, and G. Salvanesohi, “Distributed system development with ScalaLoci”, Proceedings of ACM Programming Languages, vol.2, no.OOPSLA, pp.129:1-129:30, New York, NY, USA, 2018.

[8] P. Weisanburger and G. Salvanesohi, “Multitier modules”, Proceedings of the 33rd European Conferences on Object-Oriented Programming, ECOOP, pp.3:1-3:29, Leibniz Int’l Proc. in Informatics (LIPIcs), July 15-19, 2019.

[9] P. Weisanburger, Philipp Martens, and G. Salvanesohi, “Multitier reactive programming in high performance computing”, Proceedings of the 6th ACM SIGPLAN Int’l Workshop on Reactive and Event-based Languages and Systems, REBELS, October 20-25, Athens, Greece, 2019.

[10] D. Rajchenbach-Teller and F.R. Sinot, “Opa: language-support for a sane, safe and secure web”, Proceedings of the OWASP AppSec Research, 2010.

[11] B. Reynders, F. Piessens, and D. Devriese, “Gavial: programming the web with multi-tier FRP”, Programming 2020, March 23-26, 2020.

[12] E. K. Cooper and P. Wadler, “The rpc calculus”, Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative programming, pp.231-242, Coimbra, Portugal, 2009.

[13] M. Neubauer and P. Thiemann, “From sequential programs to multi-tier applications by program transformation”, Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, pp.221-232, Long Beach, California, USA, January 2005.

[14] M. Neubauer, “Multi-tier programming”, Ph.D. Thesis, Universitt Freiburg, 2007.

[15] K. Choi and B-M. Chang, “A theory of rpc calculi for client-server model”, Journal of Functional Programming, Cambridge University Press, vol.29, no.5, pp.1-39, 2019.

[16] K. Choi, J. Cheney, S. Fowler, and S. Lindley, “A polymorphic rpc calculus”, 22th Brazilian Symp. on Formal Methods (SBMF), São Paulo, Brazil, November 27-29, 2019.

[17] E. Cooper, S. Lindley, P. Wadler, J. Yallop, “The essence of form abstraction,” In: Ramalingam G. (eds) Programming Languages and Systems, APLAS 2008, Lecture Notes in Computer Science, Springer, vol.5356, pp.205-220, Bangalore, India, December 2008.

[18] A. Chlipala, “Ur: statically-typed metaprogramming with type-level record comptation”, Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI ‘10), SIGPLAN Notices, vol.45, no.6, pp.122-133, New York, NY, USA, Jun 2010.

[19] L. Wong, “Kleisli, a functional query system”, Journal of Functional Programming, vol.10, no.1, pp.19-56, Cambridge University Press, January 2000.

[20] S. Lindley, J. Cheney, “Row-based effect types for database integration”, In Proceedings of the 8th ACM SIGPLAN workshop on Types in Language design and implementation, pp.91-102, Philadelphia Pennsylvania, USA, January 2012.

[21] The Elm Architecture, https://guide.elm-lang.org/architecture/.

[22] S. Fowler, “Model-view-update-communicate: session types meet the Elm architecture”, arXiv:1910.11108, January 2020.

[23] S. Fowler, “Typed concurrent functional programming with channels, actors,and sessions”, PhD Thesis, LFCS, University of Edinburgh, 2019.

[24] C. Elliott, P. Hudak, “Functional reactive animation”, In Proc. of the 2nd ACM SIGPLAN International Conference on Functional Programming, pp.263-273, Amsterdam, The Netherlands, August, 1997.

[25] E. K. Cooper, P. Wadler, “The RPC calculus”, Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, pp.231-242, Coimbra, Portugal, September 2009.

[26] J. Hughes, “Generalising monads to arrows”, Scinece of Computer Programming, Vol.37, No.1-3, pp.67-111, May 2000.

약 력

최 광 훈

1994 KAIST 전산학과 (공학사)
1996 KAIST 전산학과 (공학석사)
2003 KAIST 전산학과 (공학박사)
2003~2005 JAIST (박사후과정연구원)
2005~2006 Tohoku Univ. (박사후과정연구원)
2006~2010 LG전자 책임연구원
2011~2016 연세대학교(원주) 컴퓨터정보통신공학부 조교수
2016~현재 전남대학교 소프트웨어공학과 부교수
관심분야: 프로그래밍언어, 타입시스템, 컴파일러, 프로그램분석, 소프트웨어공학
Email: kwanghoon.choi@jnu.ac.kr