인터넷 스크립팅 언어의 동향 및 응용에 관한 연구

이 종 섭† · 최 영 근††

요 약

현재 웹(Web) 환경에서는 정보의 표현 수단으로, HTML(Hyper Text Markup Language)을 이용하여 문서를 표현하고 교환하고 있다. 하지만 HTML에서 제공하는 한정된 태그 집합으로는 다양한 형태의 문서 표현에 많은 제약이 따른다. 따라서 웹의 정적인 정보 표현 수단인 HTML에 컴퓨팅 환경 및 다양한 멀티미디어 정보를 동기화 하여 표현할 수 있는 스크립팅 언어 기술과의 접목이 요구된다. 즉, 기존의 웹 표현 형식에 새로운 응용 기술인 스크립팅(Scripting) 언어를 이용하여 웹을 확장시키고, 이를 통한 웹서비스의 질적인 향상을 도모할 수가 있다.

A Study on Trend and Application of Internet Scripting Language

Jong-Sup Lee† · Young-Guen Choi††

ABSTRACT

Currently in the Web(World Wide Web) environment, HTML(HyperText Markup Language) is used for information representation and exchange. But it is thought that HTML has some constraints in information representation of various kinds because of its limited tag set. And it is considered that combining the HTML, which is used for static information representation in Web environment, with Scripting language, which is usually used for multimedia information representation in a synchronized framework, can be very useful. Consequently we show the general trend of the Scripting language in Web environment and show the possibility of HTML and Scripting language amalgamation for Web service improvement.

1. 서 론

정보통신의 발전으로 초고속 정보 통신망에 근거한 다양한 응용 서비스가 제공되고 있다. 대표적으로 1989년 Tim Berners-Lee에 의해 개발된 웹(World Wide Web)은 현재 인터넷의 80% 이상을 점유하는 대표적인 서비스로 자리 매김을 하였다[9]. 웹은 HTML(Hyper Text Markup Language)을 통해 다양한 형식의 데이터를 매우 쉽게 표현한다. 또한 모든 사용자 환경을 쉽게 접근할 수 있기 때문에 인터넷 서비스 중에서 우세한 위치를 점유하고 있다. 하지만 HTML은 단순하고 비동기적이며 일방적인 형태의 제한된 표현 방식을 제공한다. 따라서 다양한 형태의 동기적인 데이터 및 네트워크 자원을 효과적으로 교환 및 검색하기에는 한계를 가지고 있다[1]. 그러므로 개발되었거나 또는 개발중인 기존의 다양한 기술들이 웹 기술과의 접목을 통해, 정적인 텍스트 기반의 구조에서 동적인 표현 방식으로 바뀌고 있으며, 이와 더불어 웹과 관련된 기술 개발이 W3C를 중심으로 표준화가 이루어지는 추세이다. 이와 같이 웹에서 정적인 정보 표현 수단인 HTML에 컴퓨팅 환경 및 다양한 멀티미디어 정보를 동기화 하여 표현할 수 있는 스크립팅 언어 기술이 접목되기 시작하였다. 그 대표적인 예가 넷스케이프(Netscape)의

† 정 회 원 : 문경대학 사무정보과 교수
†† 정 회 원 : 광운대학교 컴퓨터과학과 교수
논문접수 : 1999년 9월 18일, 심사완료 : 1999년 11월 6일

JavaScript이며, 그 외에도 Perl, Tcl, Python, Rexx, Visual Basic 등을 들 수가 있다. 따라서 기존의 표현 형식에 새로운 응용 기술인 스크립팅(Scripting) 언어를 이용하여 웹을 확장시키고, 이를 통한 웹서비스의 질적인 향상을 이룩할 수가 있다. 본 논문에서는 차세대 웹 환경을 제공할 수 있는 스크립팅 언어 기술에 대한 동향과 기존의 프로그래밍 언어와의 차이점을 기술한다. 그리고 1998년 8월에 ECMA(European association for standardizing information and communication systems)[4]에 의해 채택된 ECMAScript의 개요 및 특징을 설명하고, ECMAScript와 관련된 응용 기술인 JavaCC[6]와 구현 제품인 FESI(Free ECMAScript Interpreter)[7] 그리고 스크립트와 관련된 응용들을 설명하고 마지막으로 결론을 맺는다.

2. 스크립팅 언어의 종류 및 비교

본 장에서는 서론에서 언급한 기존 웹 상의 정적인 정보 표현 수단인 HTML에 동적 컴퓨팅 환경과 동기화된 정보 표현이 가능한 스크립팅 언어의 개발 배경을 설명한다. 그리고 기존 언어와의 차이점 및 스크립팅 언어의 선택 요건에 대해 기술한다.

2.1 스크립팅 언어의 배경

지난 30년 동안 대부분의 주요 컴퓨팅 플랫폼(Platform)들은 다양한 시스템 프로그래밍 언어 및 스크립팅 언어(Scripting Language)들을 제공해 왔다. 초기의 스크립팅 언어는 OS/360에서 순차적인 작업 처리에 사용한 JCL(Job Control Language)이었다. 각 단계의 세부 작업은 그 시대의 시스템 프로그래밍 언어인 PL/1, FORTRAN, 또는 어셈블리 언어를 사용하였다. 1980년대에 들어와 Unix에서는 C 언어가 시스템 프로그래밍 언어로 사용되었으며, sh와 csh과 같은 UNIX의 shell 프로그램이 스크립팅 언어로 등장하게 되었다. 1990년대에 들어와 PC(Personal Computer)에서는 대표적으로 C와 C++언어가 시스템 프로그램으로 사용되었고, 마이크로소프트의 Visual Basic이 스크립팅 언어로 이용되기 시작하였다. 그리고 인터넷이 급속하게 성장되고 있는 현재는 Java가 시스템 프로그래밍의 자리를 차지하게 되었고, 스크립팅 언어로는 JavaScript, Perl, JScript Tcl, Python, Rexx, Visual Basic 등이 사용되고 있다.

〈표 1〉 웹에서 스크립팅 언어의 사용 예

기술	참조횟수	사용률(%)	점유율(%)
JavaScript (script)	300,000	1.5	91
Java (system)	30,000	0.15	9
VBscript (script)	1,000	0.005	0.3
ActiveX (system)	400	0.002	0.1

위의 <표 1>은 2천만 웹 페이지로부터 조사된 스크립팅 언어의 사용 예를 참조횟수와 사용률 그리고 다른 기술과의 상대적인 점유율을 비교한 결과를 보여주고 있다[1].

2.2 기존 언어와 스크립팅 언어의 차이점

앞서 언급한 Perl, Python, Rexx, Tcl, Visual Basic 그리고 Unix shell과 같은 스크립팅 언어는 C, C++ 또는 Java와 같은 시스템 프로그래밍 언어와 매우 다른 스타일의 작업을 수행하기 위해 설계되어진 언어이다. 따라서 언어 그 자체에 시스템 프로그래밍 언어와는 다른 근본적인 차이를 가지고 있다. 즉, 스크립팅 언어는 다른 언어(시스템 프로그래밍 언어)로 이미 작성된 콤포넌트(Component)들을 서로 결합시켜서 강력한 하나의 콤포넌트 집합을 만들 수 있도록 설계되어진 언어이다[1]. 따라서 스크립팅 언어는 콤포넌트의 특성을 확장하는데 사용되기도 하지만, 반면에 복잡한 알고리즘이나 데이터 구조의 구현에는 좀처럼 사용되지 않는다. 스크립팅 언어는 복잡한 알고리즘이나 데이터 구조가 콤포넌트로 이미 구현되어 있다고 가정한다[1]. 따라서 스크립팅 언어는 응집 언어(Glue Language) 또는 시스템 통합 언어(System Integration Language)로 불리어진다[1].

2.3 스크립트 언어의 장점

스크립팅 언어는 앞서 설명한 것처럼 시스템 프로그래밍 언어로 이미 작성된 콤포넌트들을 서로 결합시켜서 강력한 하나의 콤포넌트 집합을 만들 수 있도록 설계되어진 언어이기 때문에 다른 언어에 비해 다음과 같은 장점을 갖는다[1].

- 빠른 개발

  스크립트는 시스템 프로그래밍보다 프로그램 작성 시간이 적게 걸린다. 즉, 대부분의 프로그래머들에게 지루한 작업인 데이터 또는 메모리 관련 문제들을 스크립트 플랫폼에 위임함으로서 프로그램 작성 비

용을 줄일 수 있다.

- 유연한 결합(Flexible Gluing)

스크립팅 언어는 콤포넌트들간의 결합을 단순화시키기 위해 무형(Typeless)인 경향이 있다. 즉, 콤포넌트들을 서로 결합하여 사용하는데 특정한 제약이 없다. 또한 모든 콤포넌트들과 값들을 일정한 형식으로 표현한다. 따라서 임의의 콤포넌트나 값들이 어떤 상황에서도 사용될 수 있다. 즉, 어떤 목적을 위해 설계된 콤포넌트는 설계자가 결코 예상하지 못한 완전히 다른 목적에 사용될 수 도 있다. 예를 들어, Unix shell에서 필터(Filter) 프로그램은 바이트 스트림(byte stream)으로 입력을 받아 출력으로 바이트 스트림을 내보낸다.

예) $ w -h | grep pm | cut -c1-8

따라서 스크립팅 언어는 문자열-지향(String-Oriented) 언어라고도 하는데, 이것은 여러 다른 언어들에 비해 일정한 표현 방식을 제공하기 때문이다[1].

- 안전성

스크립트 환경은 프로그래밍 버그(Bugs)의 위험성을 배제해 준다. 예를 들면 divide-by-zero와 같이 몫을 0으로 나누는 경우 자세한 에러 메시지를 제공함으로서 프로그래머는 걱정할 필요가 없다.

- 보안성

스크립트 환경에서의 모든 호출은 반드시 코드 상에 임의의 보안 레벨을 부여할 수 있는 스크립트 환경을 통해서만 전달되어야 한다. 즉, 위험한 호출이 직접 운영체제(Operating System)로 들어가는 것을 방지할 수 있다. 대부분 스크립트 언어들은 다양한 보안 옵션들과 함께 보안 모델을 제공한다.

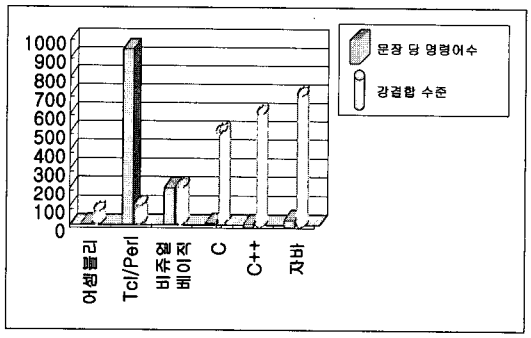
- 간결한 코드

스크립팅 언어는 컴파일러 대신에 인터프리터를 사용하기 때문에 부분적으로 시스템 프로그래밍 언어보다는 효율적이지 못하다. 반면에 기본 구성 요소인 콤포넌트들을 하드웨어에 효율적으로 사상(Mapping)시키기 위해 소비되는 컴파일 비용(Cost)에 비해 콤포넌트들을 매우 강력하면서도 쉽게 사용할 수 있다. 따라서 이러한 점에서 스크립팅 언어 선택을 고려할 수 있다.

- 기타

스크립팅 언어의 실행 속도는 일반적으로 중요 문제점은 아니다. 즉, 스크립팅 언어로 작성된 응용의 크기는 시스템 프로그래밍 언어의 응용보다 일반적으로 작기 때문에 결과적으로 스크립팅 응용의 실행 속도는 시스템 프로그래밍 언어로 구현된 콤포넌트들의 실행에 의해 좌우되는 경향이 있다. 스크립팅 언어의 단일 문장은 평균적으로 시스템 프로그래밍 언어보다 더 많은 명령어(Instruction)를 수행하기 때문에 더 고수준(High-Level)의 언어라고 볼 수 있다. 스크립팅 언어에서 한 문장은 수백 또는 수천의 기계어 명령을 수행하지만, 시스템 프로그래밍 언어에서는 대략 5개의 기계어 명령을 수행한다[1]. 이 같은 차이점은 스크립팅 언어가 인터프리터를 사용하고, 원시 명령(Primitive Operation)들이 함수성(Functionality)을 갖기 때문이다[3].

다음 (그림 1)은 언어 수준에 근거한 여러 프로그래밍 언어를 비교한 것이다. C 언어와 같은 시스템 프로그래밍 언어는 한 문장 당 5~10개 정도의 명령어 수준이고, 강결합 형(Strongly Type)인 경향이 있으며, 반면에 Tcl과 Perl 같은 스크립팅 언어는 900~1000개의 명령어 수준이며, 약결합 형(Weakly Type)인 경향이 있다[1].



(그림 1) 언어 수준(level)에 근거한 명령어 개수

2.4 스크립팅 언어의 종류

- Perl

인터넷을 통해 확인 할 수 있듯이 대부분의 서버 측 CGI(Common Gateway Interface) 스크립트는 Perl을 많이 사용한다. Perl은 80년대 후반 래리 월(Larry Wall)에 의해 처음 소개되었고, 유닉스 사용자 사이에서 시스템 관리 언어로 애용되다가, 90년대 중반을 전후해서 웹의 등장으로 범용성을 갖춘 스크립트 언어로 자리 매김을 하였다. 소스 코드나

개발 툴은 무료로 사용할 수 있다.

- JavaScript
  넷스케이프사의 Navigator 2.0 이후 버전부터 채택되어 널리 알려졌으며, 현재는 ECMA 스크립트 표준을 포함하고 있다. 현재 대부분의 클라이언트 측 웹 스크립트는 자바스크립트를 사용한다(대략 자바보다 열 배정도 많이 쓰인다)[1]. 넷스케이프와 마이크로소프트 그리고 그 밖의 다른 브라우저들은 자바 스크립트를 지원하고 있다. 또한 자바 스크립트를 변형한 자체 스크립트를 지원하는데, 한 예로 마이크로소프트사는 자사의 JScript, Nombas[8]사는 ScriptEase를 지원한다.
- Tcl/Tk
  Tcl/Tk(Tool Command Language/Graphical User Interface Toolkit)은 Sun사에서 개발했다[10]. 대부분의 시스템(도스나 윈도우즈 3.1을 제외한)에서 사용될 수 있는 간결하고 강력한 스크립트 언어이다. 또한 크로스-프랫폼(Cross-Platform) 형태의 GUI 라이브러리를 지원하는 Tk와 연동하여 더욱 강력한 기능을 제공한다. 과학적인 업무를 처리 할 때나 Unix와 관련된 작업을 처리 할 때 그리고 그 밖의 일반적인 작업이나 크로스-프랫폼 형태의 GUI를 처리할 때 많이 사용된다. 소스와 구현 툴이 무료로 제공된다.
- 비주얼 베이직(Visual Basic)
  마이크로소프트에 의해 개발된 비주얼 베이직은 GUI 응용들을 개발하는데 매우 강력한 기능을 제공한다. 오늘날 대부분의 응용들은 비주얼 베이직 언어로 많이 작성되고 있으며, RAD 스크립팅과 원리가 비슷하다. 반면에 시스템 프로그래밍 언어의 특성도 포함하고 있기 때문에 스크립팅 언어의 범주로 간주하지 않는 경향도 있다.
- 기타
  다음은 웹에 적용이 되지 않았거나 현재 연구 중인 스크립트 언어의 종류들을 보여주고 있다.

  - VBScript : JavaScript에 대한 초기 마이크로소프트사의 대응 모델이다. 그러나 매우 한정적으로 사용되었으며, 현재 MS 서버용으로 사용하기 위해 연구 중이다.
  - Rexx : Rexx는 IBM에서 개발한 절차적(Procedural) 프로그래밍 언어이며, 일반 사용자들이 워드나 숫자와 같은 데이터 형을 다양한 심볼 객체로 쉽게 사용할 수 있도록 설계되어진 언어이다. 또한 다른 언어로 작성된 프로그램이나 함수들을 호스트 환경에서 호출할 수 있는 기능을 갖고 있으며, 단순한 프레임워크에서 최소의 비용으로 강력한 문자 처리와 연산을 수행하는 프로그램을 작성할 수 있다.
  - Python : Python는 인터프리터, 인터액티브, 객체지향 프로그램 언어이며 Tcl, Perl 그리고 Java와 유사한 프로그래밍 언어이다. 고수준의 동적 데이터 형과 예외상황 (Exception) 그리고 클래스, 모듈등의 기능을 제공하며 X11, Motif, Tk, Mac, MFC 등의 라이브러리를 호출할 수 있는 인터페이스를 보유하고 있다. 또한 Python는 여러 Unix 환경 및 Dos, OS/2, Mac등의 여러 환경에서도 실행 가능하다.

이외에도 Scheme, Oberon, Dylan, S-lang등의 스크립트 언어들이 연구 중에 있다.

〈표 2〉 스크립트 언어와 시스템 프로그래밍 언어로 작성된 동일한 응용 비교

적용기술	개발사	인터프리터의 크기(KBytes)	Puzzle 프로그램 (라인 수)
스크립팅 언어			
Tcl(client)	Sun	~3000	30
Perl(server)	Free	~500	280
JavaScript(client)	Netscape	~250	160
JavaScript(server)	Nombas	~300	160
시스템 언어			
Java(client)	Sun/Netscape	~6500	2030

~ : 약(about)

3. 적절한 언어 선택

스크립팅 언어와 시스템 프로그래밍 언어는 각각 서로 다른 작업에 적합하다. 시스템 통합(Integration)과 시스템 결합(Gluing)을 위한 응용은 시스템 프로그래밍 언어 보다 스크립팅 언어로 약 5~10배 정도 빠르게 개발될 수 있다[1]. 반면에 복잡한 알고리즘이나 데이터 구조에 대해서는 시스템 프로그래밍 언어가 프로그램을 관리하기에 적합하며, 실행 속도 또한 스크립팅 언어보다 약 10~20배 빠르다[1]. 다음은 LAR(Language Applicability Rating) 공식에 따라 어떤 언어와 툴을

사용할 것인지 결정하기 위한 선택 요인들을 나타내고 있다[1].

$$LAR = \frac{wPerf \times wDynam \times wFlex \times wUbiq}{wSkillCost \times wTime \times wSys \times wToolCost}$$

- √ wPerf (Performance) : 언어 실행 속도가 빠른가?
- √ wDynam (Dynamics) : 사용자 상호작용(Interaction) 또는 다양한 데이터에 근거해 변경되는 동적 응용을 생성하는가?
- √ wFlex (Flexibility) : 결과를 변경할 수 있는 유연성(Flexibility)이 있는가?
- √ wUbiq (Ubiquity) : 사용자에게 유용한 콤포넌트와 툴을 제공하는가?
- √ wSkillCost : 해당 언어를 이용해서 개발하는데 어느 정도의 기술이 요구되는가? 또는 어느 정도 사용자에게 유용하며, 비용은 얼마인가?
- √ wTime : 이 툴을 이용해서 걸린 개발 시간은 얼마인가?
- √ wSys : 어느 정도의 기억장치, CPU 종류 그리고 어떤 OS가 이 언어를 사용하는데 필요한가?
- √ wToolCount : 언어 툴의 가격은 얼마인가? (무시될 수 있음)

LAR 공식은 앞서 기술한 기준 항목들의 가중치를 가지며, 정규공식이라기 보다 각 사용자의 목적에 맞는 적절한 프로그래밍 언어를 선택할 때 도움 줄 수 있는 하나의 도구로 이용될 수 있다. 예를 들어 일반적인 웹 페이지를 개발하려고 할 때, 기존에 사용하는 시스템 요구사항을 설정하는 것이 매우 중요한데, 그 이유는 모든 사람들이 새로운 설치작업을 하지 않고도 페이지 접근이 가능해야 하기 때문이다. 그리고 짧은 요청 시간 또한 매우 중요하다. HTML은 이 같은 경우에 매우 높은 등급을 갖는다.

반면에 대부분의 동적 프로그래밍 환경에서는 적은 기술 비용 및 개발 시간이 드는 스크립트 언어가 높은 등급을 갖게될 것이다.

4. ECMAScript의 소개

ECMAScript는 JavaScript와 Jscript등과 같은 몇몇 잘 알려진 기존의 기술(Technologies)에 근거하여 개발된 스크립트 언어이다[4]. ECMAScript는 1996년 11월에 표준 개발이 시작되었으며, 1997년 6월에 ECMA General Assembly에서 ECMA 표준 초판(First Edition)이 채택되었고, 그 후 ISO/IEC JTC 1에 제출되어 1998년 4월에 국제 표준 ISO/IEC 16262로 승인되었다. 1998년 6월에 ECMA General Assembly는 ISO/IEC 16262와 완벽한 제휴를 위해 두 번째 개정판(Second Edition)인 ECMA-262를 발표하였다. 현재 ECMAScript 언어 표준 작업은 정규 표현(Regular Expression)과 풍부한 제어 문장 그리고 개선된 스트링 조작등을 지원하기 위해 지속적인 작업이 이루어지고 있다. 또한 try/catch exception 조작과 더 포괄적인 기능(Facilities)들을 포함한 세 번째 개정판(Third Edition)을 1999년 말에 발표할 예정에 있다. 본 장에서는 ECMA의 역사와 활동 배경 및 목적에 대해 기술한다.

4.1 ECMA의 역사

1959년 여러 제조업체들에 의해 컴퓨터가 제작되어 사용이 증가되면서 프로그래밍 및 I/O 코드와 같은 운영 기술들의 표준화가 대두되기 시작했다. 비록 1960년대 전후해서 일부 국가 단체에 의해 종이 테이프와 코드 분야에 표준화 작업이 시작되었지만, 협동 연구 및 제조업체들 간에 조율은 힘들었다. 따라서 이러한 작업을 통합하는 목적으로 유럽 선두 업체들이(Compagnie des Machines Bull, IBM World Trade Europe Corporation, International Computers, Tabulators Limited) 1960년 5월 27일 브뤼셀에서 최초의 모임을 갖게 되었다. 이 제조업체 연합은 ECMA(European Computer Manufacturers Association) 구성을 위한 위원회를 만들었고, 1960년 12월에 ECMA 정식 단체가 구성되었으며, ISO/IEC(International Organization for Standardization and the International Electrotechnical Commission)의 본부를 제네바에 두고, 1961년 5월에 정식 협회가 설립되었다.

ECMA는 ISO/IEC에 의해 조직된 제네바 Round-Table Conference에서 TC97의 구성과 자체의 Working Groups를 구성했다. 1994년에 Europe-based ECMA Oganization의 국제적 활동들을 반영하기 위해 ECMA의 명칭을 ECMA(European association for standardizing information and communication systems)로 변경하게 되었다.

4.2 ECMAScript의 개요

ECMAScript는 Netscape사의 웹 스크립팅 언어인 JavaScript를 근거로 최근에 발표된 스크립팅 언어 표

준이다. 이 표준의 제정은 Netscape사의 발의권(Original Initiative) 하에서 표준화 기구인 ECMA 위원회에 의해 작성되었다. 마이크로소프트와 넷스케이프는 위원회 멤버이며 각 사의 브라우저에 ECMAScript 표준 인터프리터를 제공하기로 약속했다. 원래 ECMAScript는 동적인 웹 페이지를 위해 개발되었지만, 더 광범위한 사용목적을 위해 확장할 수 있도록 설계되었다. ECMAScript 언어 명세(Specification)에는 다음과 같이 언급하고 있다. "ECMAScript는 여러 호스트 환경을 위해 코아(Core) 스크립팅 능력을 제공할 수 있으며, 따라서 코아 스크립팅 언어는 어떤 특정한 호스트 환경에 제한되지 않는다." (ECMAScript 언어 명세 p.1,2). 이와 같이 문서에 명시된 독립적인 호스트 환경은 어떤 종류의 응용에 대해서도 ECMAScript가 훌륭한 스크립팅 언어로써의 기능을 제공한다는 것을 의미한다.

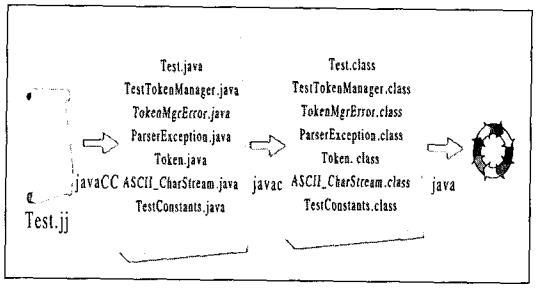
ECMAScript는 호스트 환경 내에서 객체를 조작하고 실행하기 위한 객체-기반(Object-based) 프로그래밍 언어이다. ECMAScript는 원래 브라우저에서 웹 페이지의 동적인 메카니즘을 제공하고, 웹 기반 클라이언트-서버 아키텍쳐에서 서버 측 컴퓨팅을 수행하기 위한 웹 스크립팅 언어로 설계되어졌다. 또한 여러 종류의 호스트 환경을 위한 코아 스크립팅 능력을 제공할 수 있으며, 따라서 코아 스크립팅 언어는 어떤 특정 호스트 환경과 독립적으로 명시되어 있다.

5. 관련된 응용

본 장에서는 ECMAScript와 관련된 구현 예와 Sun에서 개발한 고수준 문법(Grammar) 명세를 컴파일 하여 파서(Parser)를 자동으로 생성해내는 JavaCC와 이를 이용하여 ECMAScript를 구현한 FESI(Free ECMAScript Interpreter) 및 관련된 응용들에 대해 기술한다.

5.1 JavaCC

JavaCC(Java Compiler Compiler)[6]는 Java 언어의 파서를 생성하기 위한 JavaCC 프로젝트로부터 시작되었다. 즉, Java API 테스팅 툴인 QuickTest의 한 부분으로 Java 파서(parser)를 생성하기 위해 개발되었으며, JavaCC의 초기 버전은 1996년 6월에 JACK[5]이라는 이름으로 발표되었다. JavaCC는 하나의 텍스트 파일 안에 저장된 고수준 문법 명세(High-level Grammar Specification)를 컴파일 하므로 써 자동으로 파서를 생성해 낸다. 다음은 사용자의 고수준 문법 기술(High-level Grammar Descriptions)에 근거한 파서 생성 방법 및 응용 구성 방법에 대해 기술한다.



(그림 2) JavaCC의 실행 과정

5.1.1 JavaCC의 실행

먼저 임의의 파서를 생성하고자 하는 사용자는 JavaCC의 문법 및 어휘 규칙에 맞게 작성된 "*.jj" 확장자를 갖는 입력 파일을 작성해야한다. 다음으로 JavaCC 툴을 사용하여 실행시키면, 해당 입력 파일의 문법을 분석한 몇 개의 Java 파일이 생성된다. 이것이 *.jj 파일에 대한 파서이다. 이 파일을 javac로 컴파일 한 후, 실행한다.

5.1.2 JavaCC의 파서 문법 기술

JavaCC 파서 기술(Descriptions) 화일은 확장자가 *.jj 을 가지며, 크게 세 부분으로 구성되어있다. 즉, 옵션(Options)과 기본 클래스(Base class)부분, 렉시칼 토큰(Lexical tokens)부분 마지막으로 비-단말(Non-Terminals) 부분으로 구성된다.

<표 3>은 매우 단순한 JavaCC 문법을 보여주고 있다. 이 문법은 왼쪽 중괄호("{")의 수와 부합되는 오른쪽 중괄호("}")를 인식하고, 그 뒤에 0개 이상의 라인 종료자(Line Terminator)와 파일의 끝을 인식하는 문법을 기술한 것이다. <표 3>은 JavaCC에서 제공하는 디폴트 값으로 설정된 모든 옵션들을 보여준다. 옵션 설정은 실제로 요구되지는 않지만, 필요에 따라 설정될 수 도 있다. 자세한 옵션 설정은 JavaCC 문서를 참조하길 바란다[6].

다음으로 "PARSER_BEGIN(Test)"와 "PARSER_END(Test)"로 싸여진 Java 컴파일 단위가 나온다. 이때 주의할 사항은 클래스의 이름(예 : Test) 즉, PARSER_BEGIN과 PARSER_END의 인자(Argument)와 Java 클

〈표 3〉 간단한 JavaCC 문법

```
options {
  LOOKAHEAD = 1;
  CHOICE_AMBIGUITY_CHECK = 2;
  OTHER_AMBIGUITY_CHECK = 1;
  STATIC = true;
  DEBUG_PARSER = false;
  DEBUG_LOOKAHEAD = false;
  DEBUG_TOKEN_MANAGER = false;
  ERROR_REPORTING = true;
  JAVA_UNICODE_ESCAPE = false;
  UNICODE_INPUT = false;
  IGNORE_CASE = false;
  USER_TOKEN_MANAGER = false;
  USER_CHAR_STREAM = false;
  BUILD_PARSER = true;
  BUILD_TOKEN_MANAGER = true;
  SANITY_CHECK = true;
  FORCE_LA_CHECK = false;
}

PARSER_BEGIN(Test)

public class Test {

  public static void main(String args[]) throws ParseException {
    Test parser = new Test(System.in);
    parser.Input();
    System.out.println("jslee");
  }

}
PARSER_END(Test)

void Input() :
{}
{
  MatchedBraces() ("\n" | "\r")* <EOF>
}

void MatchedBraces() :
{}
{
  "{" ( MatchedBraces() )? "}"
}
```

래스의 이름이 동일해야 한다는 것이다. 이 클래스의 이름은 파서 생성기에 의해 생성된 Java 파일(파서)에 Prefix로 사용된다. 그리고 파서의 코드는 다음과 같이 Java 클래스의 종료 중괄호("}") 바로 앞에 삽입되어 진다.

```
...
class Test ...{
  ...
  // 생성된 파서는 이곳에 삽입된다.
}
...
```

Test 클래스 안에 static으로 선언된 main 메소드는 명령 라인에서 java 인터프리터에 의해 호출되는 클래스이다. 이 main 메소드는 단순히 입력 스트림(System.in)으로 부터 새로운 파서를 인스턴스화 하고, Input 메소드를 호출한다.

Input 메소드는 작성한 문법 <표 3>에서처럼 비-단말(Non-Terminal)이며, EBNF(Extended Backus-Naur Form)의 형식으로 정의되어 있다. 이 EBNF는 문맥 자유 문법(Context-Free Grammars)을 명시하기 위한 방법이다. 이 명세는 왼쪽에 터미널(Terminal)이 나타나고 생성 심볼(Production Symbol)은 "::="으로 표기하며, 오른쪽에 하나이상의 생성(Production)이 나타날 수 있다. 간단한 표기식은 다음과 같다.

```
Keyword ::= "if" | "then" | "else"
```

JavaCC에서는 왼쪽 부분 즉, 터미널을 메소드로 대치시킴으로서 문법 확장이 가능하다. 또한 정규 표현(Regular Expression) 또는 다른 비-단말들에 의해 여러 형태의 확장도 가능하다. JavaCC의 문법에서 어휘 토큰(Lexical Token)은 단순한 문자열(예 : "{", "}", "\n", "\r")이거나 더 복합된(Complex) 정규 표현이 나올 수도 있다. 위의 예에서 정규 표현 "<EOF>"는 화일의 끝으로 인식된다. 모든 복합 정규 표현은 "<"과 ">"안에 기술된다. 각 생성(Production)은 콜론(" : ")에 의해 왼쪽 측 비-단말을 정의하며, 첫 번째 중괄호 안에 전역 변수 선언과 문장들을 포함할 수 있다. 위의 예제에서는 "{}"으로 어떤 선언도 없다. 첫 번째 생성인 비-단말 "Input"은 "MatchedBraces" 비-단말로 확장되며, 그 뒤에 0개 이상의 라인 종료자인 "\n"과 "\r"이 나올 수 있고, 마지막에 파일의 끝("<EOF>")을 나타내고 있다. 두 번째 생성인 비-단말 "MatchedBraces"는 토큰 "{"과 "}" 사이에 "MatchedBraces"를 선택적으로 다시 되부르는 생성이다. 이때 "[...]"은 옵션을 의미하며, "(...)?"으로 표기할 수도 있다. 다음 <표 4>는 <표 3> 문법에 맞는 입력 예와 잘못된 예를 보이고 있다.

〈표 4〉 옳은 예와 틀린 예

옳은 예	틀린 예
{}	{ }
{{{}}}<enter>	{{}<enter>}
{} 	{}{}
{{}}	{{}{}}

5.1.3 JavaCC의 특징

JavaCC는 순수 Java로 작성된 Java 파서 생성기이다. 이것은 순수 Java 코드로 작성되었기 때문에 JavaCC에 의해 생성된 파서는 여러 Java 플랫폼에서 실행 가능하며, 또한 Java 1.0.2와 Java 1.1의 모든 문법을 포함하고 있다[5]. 다음은 JavaCC의 특징을 설명하고 있다.

- Top-Down : JavaCC는 YACC의해 생성되는 Bottom-Up 파서와는 반대로 Top-Down 파서를 생성한다. 비록 Left-Recursion이 가능하지 않지만, 더 범용적인 문법의 사용을 허용한다. Top-Down 파서는 디버깅(Debugging)이 쉽고, 문법 내의 어떤 비-단말(Non-Terminal)도 파싱할 수 있는 능력과 같은 여러 장점을 가진다.
- 한 파일 안에 어휘와 문법 명세 기술 : 어휘는 스트링과 그 외의 모든 것을 정규 표현으로 명시하며, 문법 명세(BNF) 또한 같은 화일 안에 함께 작성한다. 이것은 사용자에게 문법을 이해하고 유지 및 보수하는데 큰 도움을 준다.
- 세밀한 옵션 : JavaCC는 생성된 파서들의 동작을 제어할 수 있는 여러 옵션을 제공한다. 한 예로, 입력 스트림 상에서 다양한 Unicode 처리 및 다수의 모호한 토큰을 검사할 수 있는 여러 옵션들을 제공한다.
- 100% 순수 Java : JavaCC는 Java가 실행되는 모든 플랫폼 즉, 버전 1.0.2 이후 버전에서 실행된다. Java의 "Write Once, Run Everywhere"라는 모토에 따라 특정한 노력없이 어떤 머쉰에서도 사용될 수 있다.
- 문법 적용범위 체크 : Java 테스트 적용 툴인 JavaScope와 더불어 Java 문법의 적용 범위가 생성(Production)이나 확장에 근거하여 검증될 수 있다.
- 범용적 : JavaCC의 어휘 분석은 완벽하게 Unicode 입력을 처리할 수 있으며, 어휘 명세는 임의의 Unicode 문자를 포함시킬 수 도 있다. 이 같은 기능은 Java 식별자(Identifier)와 같은 언어 요소들을 기술할 수 있다.
- Syntatic & Semantic LOOKAHEAD 명세 : 기본적으로 JavaCC는 LL(1) 파서를 생성하지만 LL(1)이 아닌 일부의 문법을 포함할 수도 있다. JavaCC는 지역적으로 shift-shift 모호성을 해결하기 위해 Syntactic 및 Semantic Lookahead의 기능을 제공한다. 즉, 파서는 해당 시점에서만 LL(k)지만 더 나은 수행을 위해 다른 모든 곳에서는 LL(1)이다. shift-reduce와 reduce-reduce의 충돌은 top-down 파서에서 문제가 되지 않는다.
- 확장 BNF 명세 허용 : JavaCC는 다음과 같이 확장된 BNF 명세를 허용할 수 있다. 어휘와 문법 명세 안에서 (A)*와 (A)+등의 확장된 BNF는 임의 확장시에 left-recursion에 대한 가능성을 감소시킨다. 사실 확장된 BNF은 A::=y(x)*와 A::=Ax | y에서처럼 가독성(Readability)이 높다.
- Lexical 상태 & 분석 : 어휘 명세는 전체 어휘 명세를 위한 전역 레벨에서 또는 토큰 처리 단계에서 대/소문자 구분 없이 토큰을 정의할 수 있다.
- 확장 디버깅 능력 : DEBUG_PARSER, DEBUG_LOOKAHEAD 그리고 DEBUG_TOKEN_MANAGER 등의 옵션을 사용해서 상세한 파싱 분석 및 토큰 처리가 가능하다.
- 특정 토큰(Special Tokens) : 어휘 명세에서 특정 토큰과 같이 정의된 토큰은 파싱 중에 무시(Ignore)되지만, 이 토큰들은 툴에 의해 유용하게 처리되어 진다. 이 같은 응용은 주석(Comments)처리에 유용하다.
- 훌륭한 에러 보고 : JavaCC 에러 리포팅은 파서 생성기들 중에서 가장 우수하다. 파서를 생성한 JavaCC는 완벽한 진단 정보를 가지고 파서 에러의 위치를 정확하게 지시할 수 있다.
- Tree 구성 전처리기 : JavaCC는 최근에 많은 이용되고 있는 트리 구성 전처리기인 JJTREE를 포함하고 있다.

5.2 FESI

FESI[7]는 ECMAScript 언어를 완벽하게 구현한 인터프리터이며, JavaScript 버젼 1.1과 JScript Core 부분과 매우 유사하다. 또한 몇몇의 확장(Extension)들이 ECMAScript 프로그램으로부터 기본 I/O, 화일 I/O, Java 객체에 대한 접근, DB 접근 그리고 탐색에 근거한 정규 표현 등을 제공하기 위해 적재(Load)될 수 있으며, 클래스와 빈(Beans)의 동적 로딩을 포함할 수도 있다.

FESI는 Java 응용을 위한 매크로(Macro) 언어와 같이 ECMAScript를 사용할 수 있도록 Java 팩키지(Package)의 집합으로 구성되어 있으며, 상호작용 인터프리터(Interactive Interpreter)를 포함하고 있다. FESI의 인터프리터는 배치 모드(Batch Mode)와 상호작용 모드(Interactive Mode)로 실행된다. 소스 코드의 확장자는 "js", "es" 그리고 "esw"를 인식한다.

5.2.1 FESI의 Java 라이브러리

Java 라이브러리는 ECMAScript 인터프리터의 능력을 확장하기 위해 사용되며, Java 프로그램으로 인터프리터를 생성하고 스크립트를 실행하며, ECMAScript 객체의 프로퍼티(Property)를 정의하고 사용할 수 있도

록 해준다. 이것은 Netscape의 JSObject와 JSException 클래스와 호환성이 매우 높기 때문에, 공통된 코드 개발을 단순화하는데 도움을 줄 수 있다. 라이브러리는 매우 단순하며, 단지 인터프리터의 기본 기능을 접근하는데만 사용된다. 따라서 인터프리터의 내부 구조와는 완벽하게 독립되어 있기 때문에 FESI의 변경에 의한 영향을 받지 않는다. 자세한 내용은 FESI 문서를 참고하길 바란다[7]. 또한 FESI에서는 모든 Java 클래스를 사용자가 직접 접근할 수 있도록 상호작용 인터프리터에 의해 JavaAccess 확장이 항상 적재된다. 따라서 Netscape의 "꽥키지"들과 완벽하게 호환될 수 있다.

5.2.2 간단한 예제와 관련 응용

<표 5>는 ECMAScript 예제로 시스템 관련 정보를 화면에 출력하는 프로그램이다.

〈표 5〉 간단한 ECMAScript 프로그램

```
// properties.es
//
// This example demonstrates how the system properties
// can be simply listed with a short EcmaScript
// Requires: BasicIO, JavaAccess (standard in the interpreter)
// Get the properties

properties = java.lang.System.getProperties();

// List them, using a java enumerators

for (key in properties.keys()) {
    value = properties.get(key);
    writeln ("Property '" , key, " = '", value, "'");
}
```

다음은 ECMAScript와 관련된 유용한 응용 및 제품들을 나열하고 있다.

- mpTOOLS : mpTools의 mpEdit는 color highlighting이 가능한 멀티플랫폼(Multiplatform) ECMAScript 에디터를 제공한다[http://members.tripod.com/~mpTOOLS/index.html].
- Jeremie : 이 사이트는 ECMAScript와 관련된 XML의 정보들을 보유하고 있다[http://www.jeremie.com].
- Caucho Technology : JavaScript의 컴파일러 및 ECMAScript 컴파일러와 관련된 사이트이다[http://www.caucho.com/index.html].
- PYTHON : 스크립팅 언어인 Python에 대한 정보 및 ECMAScript와 관련된 정보를 제공한다[http://www.python.org/jpython/].

6. 결 론

1960년대 이후 대부분의 주요 컴퓨팅 플랫폼들은 다양한 시스템 프로그래밍 언어 및 스크립팅 언어(Scripting Language)들을 제공해 왔으며, 따라서 스크립팅 언어와 시스템 프로그래밍 언어는 상호 보완적으로 발전해 왔다. 현재의 콤포넌트 프레임워크(Framework)에서는 시스템 프로그래밍 언어로 작성된 콤포넌트를 스크립팅 언어로 서로 결합하여 더 큰 생산성을 제공하고 있다. 하지만 스크립팅 언어는 시스템 프로그래밍 언어에 비해 상대적으로 실행 속도 및 자료형 검사(Type Check) 부분이 다소 약하다. 반면에 소프트웨어 재사용과 높은 생산성을 제공한다는 잇점을 제공한다. 그리고 다음과 같은 최근의 경향들, 즉 컴퓨터 처리능력의 향상, 다양한 기능을 포함하는 스크립팅 언어의 출현 그리고 GUI와 콤포넌트 아키텍처의 중요성 증가 및 인터넷의 급속한 성장과 같은 요인들은 스크립트 언어의 응용성(Applicability) 및 사용을 급격하게 증가시키고 있는 요인들로 볼 수 있다. 따라서 이 같은 경향들로 볼 때, 웹 컴퓨팅 분야에서 기존의 다양한 기술과 더불어 스크립팅 기술은 앞으로 새로운 분산 프로그래밍 패러다임으로 자리를 굳히게 될 것이다.

참 고 문 헌

[1] John K. Ousterhout, "Scripting : Higher Level Programming for the 21st Century," IEEE Computer magazine, March 1998.

[2] C. Jones, "Programming Languages Table, Release 8.2," March 1996, http://www.spr.com/.

[3] J. Ousterhout, Additional Information for Scripting White Paper, http://www.scriptics.com/people/john.ousterhout/scriptextra.html.

[4] http://www.ecma.ch/.

[5] http://www.javaworld.com/javaworld/jw-12-1996/jw-12-jack.html.

[6] http://www.suntest.com/JavaCC/index.html.

[7] http://home.worldcom.ch/~jmlugrin/fesi/index.html.

[8] http://www.nombas.com/us/tellme/script.html.

[9] W3C : World Wide Web Consortium, Leading the Web to its Full Potential, 1998.

[10] http://www.scriptics.com/.

이 종 섭

e-mail : jslee@munkyung.ac.kr

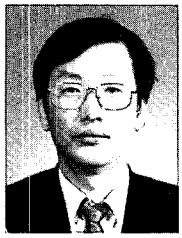
1990년 호서대학교 전자계산학과 이학사

1993년 광운대학교 대학원 전자계산학과 이학석사

1997년 광운대학교 대학원 전자계산학과 박사 수료

1997년~현재 문경대학 사무정보과 전임강사

관심분야 : 객체 지향 분산 컴퓨팅, 객체 지향 프로그래밍 언어, 웹 프로그래밍



최 영 근

E-mail : ygchoi@cs.kwangwoon..ac.kr

1980년 서울대학교 수학 교육과 이학사

1982년 서울대학교 계산통계학과 이학 석사

1989년 서울대학교 계산통계학과 이학 박사

1983년~현재 광운대학교 컴퓨터과학과 교수

1997년~1998년 미시간 주립대 객원교수

1998년~현재 광운대학교 전산정보원 원장

관심분야 : 객체 지향 분산 컴퓨팅, 병렬 프로그래밍 언어, 객체 지향 프로그래밍 언어