제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

객체 지향 메시지 기반 인터프리터

김주원*, 최성운*
*명지대학교 컴퓨터공학과
e-mail : sevensky@mju.ac.kr

Message Driven Interpreter Based Object Oriented

Joo-Won Kim*, Sung-Woon Choi*
*Dept. of Computer Engineering, Myong-Ji University

요 약

스크립트 언어는 클라이언트 시스템 개발 시 화면구성 컴포넌트간의 상호작용을 정의하기 위해 사용된다. 기존의 스크립트 언어는 접착코드(Glue Code)나 이진코드(Binary Code) 방식을 기반으로 컴포넌트 상호작용을 정의하므로 시스템의 변경에 유연하게 대처하지 못하는 단점을 가지고 있다. 본 논문에서 제시하는 FlexScript 는 스트링(String)기반의 메시지 전달 방법으로 컴포넌트 상호작용을 정의하는 스크립트 언어 시스템이다. 이러한 메시지 시스템 기반의 FlexScript 는 컴포넌트 및 스크립트 언어의 변화에 대하여 확장성과 유연성을 제공한다.

1. 서론

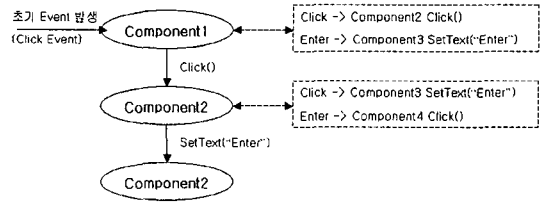
컴퓨터 프로그래밍에서 스크립트(Script)란 컴퓨터 프로세서가 아닌 다른 프로그램에 의해 번역되거나 수행되는 프로그램이나 명령어들의 나열을 의미한다. 스크립트를 작성하기 위한 스크립트 언어는 1960 년대 등장하였으며, 시스템의 개발생산성 향상을 위한 컴포넌트 기술의 등장과 함께 빠른 속도로 발전되어 왔다.[6] 컴포넌트 기반 시스템의 개발은 컴포넌트 및 컴포넌트간의 상호작용을 프로그래밍함으로써 이루어지는데 컴포넌트간의 상호작용은 컴포넌트가 제공하는 인터페이스를 통해 가능하다. 스크립트 언어를 사용하면 인터페이스를 통한 컴포넌트의 상호작용을 보다 용이하게 정의할 수가 있다.[3]

일반적인 스크립트 언어 시스템에서는 스크립트 언어와 컴포넌트 간의 상호작용을 위한 연결 방법으로 접착코드(Glue Code)나 이진코드(Binary Code)를 사용한다.[7] 이러한 접착코드나 이진코드는 컴포넌트 시스템과 스크립트 언어간의 확장성과 유연성을 떨어뜨리게 한다. 본 논문에서는 스크립트 언어와 컴포넌트간의 상호작용을 접착코드나 이진코드 없이 스트링(String) 기반의 메시지(Message) 전달에 의해 가능하도록 하는 스크립트 언어 시스템 FlexScript 을 제안한다.

2. 메시지 트리거링 시스템

본 논문에서 제안하는 FlexScript 언어는 메시지 트리거링 시스템 기반으로 개발된 컴포넌트에서 구동되는 스크립트 언어이다. 이 메시지 트리거링 시스템은 그림 1 과 같이 특정 컴포넌트에 초기 이벤트가 발생할 경우, 그 컴포넌트가 해당 이벤트를 수행한다. 그리고 나서 자신의 이벤트 메시지 맵(Event Message Map)에 정의된 연관 컴포넌트에 이벤트 메시지를 전달하고 해당 컴포넌트가 그 이벤트를 발생시키는 방식으로 구성된다.

그림 1 메시지 트리거링 시스템



3. FlexScript 언어

3.1 FlexScript 언어의 구성

FlexScript 언어에 의해 쓰여지는 구분은 다음과 같은 내용들을 기술한다.

- 이벤트를 발생시키는 컴포넌트와 그것의 이벤트내용
- 컴포넌트 간 상호작용의 순서적인 연관관계

- 기술된 컴포넌트의 이벤트들이 발생하는 일련의 과정을 하나의 집합으로 나타낼 수 있는 시나리오
- 작업의 효율성을 위해 하나의 시나리오내에서 다른 시나리오를 부를 수 있도록 부시나리오(SubScenario)
- 이벤트들과 이벤트들의 흐름을 제어하기 위한 조건문, 반복문, 논리연산문, 산술연산문, 배정문 등

각각의 구문은 구문에 맞는 특정 키워드를 가지게 되며, 표현식(Expression), 컴포넌트, 이벤트 등을 구성요소로 가지게 된다. 표 1 은 각 구문과 구성요소, 키워드를 나타낸다.

표 1 구문에 따른 구성요소와 키워드

구문	구성요소	키워드
시나리오문	모든 구문 (시나리오 구문 제외)	SCENARIO
이벤트문	컴포넌트 이벤트 부시나리오	SUBSCENARIO
배정문	표현식	
조건문	표현식 모든 구문 (시나리오 구문 제외)	IF, THEN, ELSE
반복문	표현식 모든 구문 (시나리오 구문 제외)	WHILE

표현식은 피연산자와 연산자로 구성되며 연산자는 표 2 에서와 같이 산술 연산자, 관계 연산자, 그리고 논리 연산자로 나눌 수 있다. 그리고 각 연산자의 우선순위는 표 3 과 같다.

표 2 연산자의 표기법

형태	표기법
산술 연산자	+, -, *, /, %
관계 연산자	<, <=(=<), >=(=>), >, ==, !=
논리 연산자	!, &&, ||

표 3 연산자의 우선순위

우선순위	구성요소	비고
1	()	
2	+, -	부호
3	*, /, %	
4	+, -	덧셈, 뺄셈
5	<, <=, >=, >, ==, !=	
6	!	
7	AND, OR	
8	=	배정

3.2 FlexScript 언어의 문법

FlexScript 에서는 컴포넌트간의 이벤트 뿐만 아니라 이들의 일련의 작업흐름도 기술한다. 이에 필요한 구문이 시나리오 구문과 이벤트 구문이다.

시나리오 구문은 컴포넌트 이벤트들의 일련의 작업과정의 집합을 나타내는 구문이다. 특정 시나리오 이름의 집합 형태로 이벤트 구문을 열거한다.

시나리오 구문의 형태는 다음과 같다.

```
SCENARIO 시나리오_이름
{
        이벤트 구문
        이벤트 구문
        ...
}
```

이벤트 구문은 컴포넌트의 이벤트를 기술하는 컴포넌트 기술과 제어나 연산에 관련된 엔진 이벤트를 기술하는 제어 기술로 나뉜다.

컴포넌트 기술은 컴포넌트들의 이벤트를 기술함으로써 이루어진다.

```
부모컴포넌트 . 자식컴포넌트 . 이벤트 ;
Ex) SCR01.CTRL01.Click();
```

이러한 컴포넌트들의 이벤트를 차례로 기술함으로써 이벤트들의 순차적인 연관관계를 나타낸다.

```
상위컴포넌트 . 하위컴포넌트 . 이벤트 ;
상위컴포넌트 . 하위컴포넌트 . 이벤트 ;
Ex) SCR01.CTRL01.Click(); ----------(1)
    SCR01.CTRL02.Click(); ----------(2)
```

여기서 (1)을 기준으로 볼 때, (2)는 (1)의 컴포넌트가 해당 이벤트를 수행한 다음 수행할 컴포넌트의 이벤트이며, (1)의 컴포넌트가 이벤트 수행 후 전달한 메시지를 받을 컴포넌트이다.

(2)를 기준으로 볼 때, (1)은 (2)에게 이벤트를 수행하라고 메시지를 전달하는 컴포넌트가 된다.

따라서, (1)의 컴포넌트는 Click() 이벤트를 수행한 후 (2)의 컴포넌트에게 Click()을 수행하도록 메시지를 전달하여 (2)의 컴포넌트는 메시지를 받고 Click()을 수행한다.

제어 기술은 이벤트 제어에 있어 필요한 변수, 연산문, 배정문, 제어문, 반복문 등의 내용을 기술한다.

변수 : 변수가 가질 수 있는 값은 정수값, 실수값, 문자열값이며, 변수의 형태는 타입리스(Typeless) 형태로 배정된 값에 따라 형태를 실행시간(Runtime)에 결정한다.

연산문 : 연산문은 연산자로 표 2 에 열거된 연산자를 가지며, 연산자간의 우선순위는 표 3 과 같다. 피연산자는 변수와, 정수값, 실수값, 문장열값, 컴포넌트의 이벤트를 가질 수 있다.

배정문 : 배정문은 '=' 연산자에 의해 표기된다.

제어문 : 제어문의 시작은 'IF' 키워드로 시작하여 표현식을 비교한 다음 'THEN'과 'ELSE'로 분기되어 '{'과 '}'사이의 구문들이 실행된다.

반복문 : 반복문은 'WHILE' 키워드로 시작하여 표현식을 비교한 후 '{' 와 '}' 사이의 구문들을 실행하고 반복된다.

FlexScript 로 작성된 스크립트는 시나리오 구문의 연속적인 나열로 구성된다. 각 구문의 문법은 표 4 와 같으며 구분자(Delimiter)로는 ';'을 사용한다.

표 4 스크립트 언어의 문법

구문	문법
시나리오문	SCENARIO scenario_name { Statements }
이벤트문	Component . Component . Event SUBSCENARIO scenario_name
배정문	Variable_name = expression
조건문	IF (expression) THEN { Statements } IF (expression) THEN { Statements } ELSE { Statements }
반복문	WHILE (expression) { Statements }
주석문	/* ... */

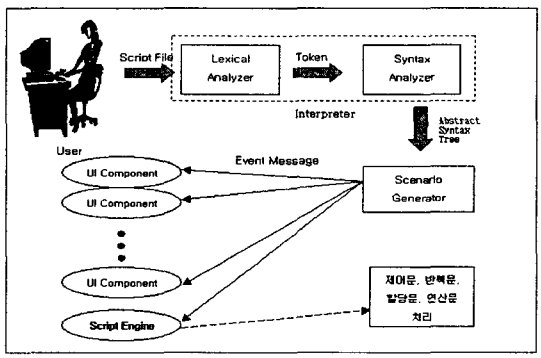
4. FlexScript 시스템의 개발

4.1 FlexScript 시스템의 구조

FlexScript 시스템은 크게 사용자가 작성한 스크립트 파일(File)을 해석하는 인터프리터와 해석된 결과를 이용하여 메시지를 생성하고 컴포넌트에 보내는 메시지 생성기(Message Generator)와 제어, 반복, 배정, 연산 등의 구문을 메시지에 의해 실제 처리하는 컴포넌트인 FlexScript 엔진(Engine)으로 나눌 수 있다.

그림 2 는 이러한 시스템의 구조를 나타낸다.

그림 2 FlexScript 시스템의 구조



4.2 인터프리터와 추상 구문 트리

4.2.1 인터프리터

인터프리터는 어휘 분석기(Lexical Analyzer)와 구문 분석기(Syntax Analyzer)로 구성되어 있으며[2], 사용자가 작성한 스크립트 파일을 읽어 이벤트 생성기에서 사용할 트리 형태(Tree Type)의 추상구문트리(Abstract Syntax Tree)를 생성한다. 추상 구문 트리는 다음 단계에서 꼭 필요한 의미정보만을 가지는 데이터 구조이다.[5]

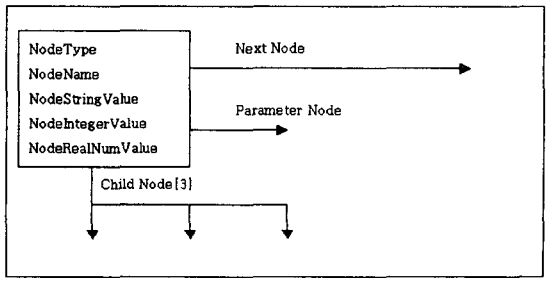
인터프리터의 수행은 어휘 분석 단계(Lexical Analysis Phase)와 구문 분석 단계(Syntax Analysis Phase) 2 단계로 이루어진다. 어휘 분석 단계에서는 스크립트 파일내의 문자열로부터 토큰(Token)과 키워드(Keyword)를 추출하고 각 토큰의 형태에 따라 형(Type)을 결정하여 구문 분석 단계로 결과를 전달한다.

구문 분석 단계에서는 어휘 분석 단계에서 추출된 토큰들을 사용하여 문법에 맞는지 확인하고, 구문형(Statement Type)을 결정하여 추상 구문 트리를 만든다.

4.2.2 추상 구문 트리

스크립트 언어를 해석한 후의 정보는 트리 형태의 추상 구문 트리로 저장된다. 이 트리는 구문 분석기의 수행 결과물이다. 트리를 구성하는 노드는 그림 3 과 같다

그림 3 추상 구문 트리의 노드



노드의 구성요소중 NodeType 은 이 노드가 어떤 내용을 담고 있는지 형태를 저장한다.

구문 노드일 경우 AssignmentStmtK, ScenrioStmtK, EventStmtK, IfStmtK, WhileStmtK 중 하나로 결정된다.

연산자의 경우 AddK, MinusK, TimesK, DivK, ModK, NeK, GeK, LeK, LtK, GtK, OrK, AndK 중 하나로 결정된다.

피연산자나 이벤트일 경우 IdK, IntNumK, RealNumK, StringK, EventK 중 하나로 결정되어 진다.

NodeStringValue, NodeIntegerValue, NodeRealNumValue 등은 각각에 맞는 형의 데이타가 저장된다. NodeName 은 NodeType 이 IdK 나 EventK 일 경우 변수(Variable)의 이름이나 이벤트의 이름이 저장된다.

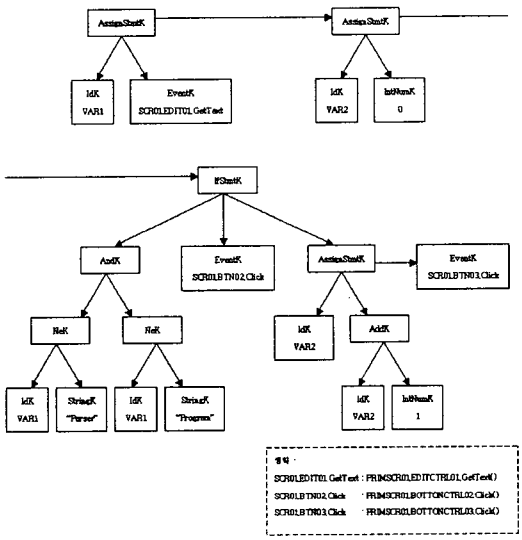
NextNode, ParameterNode, ChildNode 들은 다른 노드를 가리키는 포인터(Pointer)이며, NextNode 는 문법상 구분자로 끝난 후 다음 구문이나 파라메터 정보에서 다음 파라메터를 가리킨다. ParameterNode 는 노드가 EventK 나 EventStmtK 일 경우 이벤트에 포함된 파라메터의 첫번째를 가리킨다. ChildNode 는 구문내의 구문이나 구문내의 표현식 또는 연산문에서의 피연산자들을 가리키며 최대 3 개를 가질 수 있다.

예를 들어, 다음과 같은 구문이 스크립트 파일로 들어온다면 노드를 이용한 추상 구문 트리는 그림 4 과 같이 만들어진다.

```
VAR1 = PRIMSCR01.EDITCTRL01.GetText();
VAR2 = 0;
IF ( VAR1 != "Parser" && VAR1 != "Program" )
{
```

```
    PRIMSCR01.BUTTONCTRL02.Click();
}
ELSE
{
    VAR2 = VAR2 + 1 ;
    PRIMSCR01.BUTTONCTRL03.Click();
}
```

그림 4 추상 구문 트리의 예



4.3 메시지 생성기

메시지 생성기는 만들어진 트리를 이용하여 각 컴포넌트에게 이벤트 메시지들을 전달하여 컴포넌트가 이벤트 메시지 맵을 가질 수 있도록 하는 작업을 수행한다.

컴포넌트 1 의 이벤트 1 이 수행되었을 때 컴포넌트 2 의 이벤트 2 가 수행되어야 할 경우 컴포넌트 1 의 이벤트 1 은 수행메시지(Execution Message)로 작성 되며 컴포넌트 2 의 이벤트 2 는 전달메시지(Delivery Message)로 작성된다. 그리고 이렇게 작성된 수행메시지와 전달메시지는 시나리오 이름과 메시지의 고유한 메시지아이디(MessageID)를 넣어 수행메시지에 해당하는 컴포넌트의 이벤트 메시지 맵에 저장된다.

이러한 메시지는 크게 2 가지의 형태로 나눌 수 있다. 하나는 일반형 메시지이며 하나의 컴포넌트에 하나의 메시지가 전달될 수 있다. 다른 하나는 분기형 메시지로 하나의 이벤트에 2 가지 방향 중 하나로 메시지가 전달될 수 있다. 이는 스크립트 엔진 컴포넌트로 메시지가 전달되어지고 수행되어진다.

다음은 일반형 메시지의 형태이다.

- 수행메시지 :

  메시지 ID ; 컴포넌트이름 ; 이벤트이름 ; 파라메터갯수 ;(파라메터형 ; 파라메터값)*

- 전달메시지 :

  컴포넌트이름 ; 이벤트이름 ; 파라메터갯수 ; (파라메터형 ; 파라메터값)*

  (*는 파라메터갯수 만큼 반복된다.)

  Ex)

  SCR01.CTRL01.Click();
  SCR01.CTRL02.Click();

의 경우 시나리오 이름이 SNR01 라면 만들어지는 메시지는 다음과 같다

수행메시지 :

Message01;SNR01;SCR01.CTRL01;Click;0

전달메시지 :

Message02;SNR01;SCR01.CTRL02;Click;0

이다.

분기형 메시지는 일반형 메시지에 전달메시지가 하나더 추가된 형태이다. 수행메시지의 결과에 따라 분기가 되며 수행결과가 참일 경우 앞의 전달메시지가 전달되며 거짓일 경우 뒤의 전달메시지가 전달된다.

4.4 스크립트 엔진 컴포넌트

일반 이벤트문에 관한 것이 아니라, 이를 제어하는 조건문, 반복문, 배정문등에 관련된 이벤트는 엔진 컴포넌트에 메시지가 전달되며, 엔진 컴포넌트에서 처리하게 된다.

5. 결론

일반적으로 스크립트 언어와 컴포넌트를 연결하는 접착코드나 이진코드는 스크립트 언어와 컴포넌트에 의존적이다. 본 논문에서는 컴포넌트나 스크립트 언어에 대한 의존성을 줄이기 위해 개발, 구현된 스크립트 언어 시스템인 FlexScript 시스템을 제시하였다.

본 시스템은 스크립트 언어를 스트링 기반의 메시지로 전환하여 컴포넌트가 메시지 핸들러를 통해서 메시지를 전달하게 하였다.

스크립트 언어가 같은 추상 구문 트리를 만들 경우 스크립트 언어의 형태가 시스템 전체에 영향을 주지 않으며 컴포넌트 역시 같은 방식의 메시지 핸들러만 있으면 컴포넌트의 추가, 변경이 시스템에 영향을 주지 않는다.

그리고, 컴포넌트가 네트워크상에 분산되어있는 분산시스템에서 분산된 컴포넌트에 메시지만 전달함으로써 전체 스크립트의 내용이 반영될 수 있으므로 유용하게 사용할 수 있을 것이다.

참고문헌

[1] John R. Levine, Tony Mason & Doug Brown. "lex & yacc", 2rd Ed. O'Reilly & Associates, 1992

[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. "Compilers Principles, Techniques, and Tools", Addison Wesley, 1986

[3] John K. Ousterhout. "Scripting : Higher-Level Programming for the 21st Century, IEEE, 1998

[4] Samuel N. Kamin. "Programming Languages An Interpreter-Based Approach". Addospm Wesley, 1990

[5] Se-Man Oh. "컴파일러 입문", 정익사,2000

[6] http://www.tcl.tk/doc/scriptHistory.html

[7] http://www.doc.ic.ac.uk/~np2/patterns/scripting/