플러그인 언어로 확장 가능한 프로그래밍 언어
(An Extensible Programming Language for Plugin Features)

최 종 명 † 유 재 우 ††
(Jong-Myung Choi) (Chae-Woo Yoo)

요 약 현대적인 소프트웨어들은 모듈성과 확장성을 강조하고 있으며, 프로그래밍 언어와 컴파일러에서도 확장성에 관한 연구들이 수행되고 있다. 본 논문에서 소개하는 Argos 언어는 플러그인 언어라는 개념을 이용해서 언어와 컴파일러를 확장할 수 있는 방법을 제공한다. Argos에서 플러그인 언어는 클래스의 메소드를 정의하기 위해서 사용되며, 플러그인 언어 처리기는 동적으로 추가 및 교체될 수 있는 특징을 가지고 있다. Argos에서 플러그인 언어는 멀티패러다임 프로그래밍과 도메인 특정 언어를 지원하기 위해서 사용될 수 있다.

키워드 : 플러그인, 확장 가능한 언어, 멀티패러다임, 도메인 특정 언어

Abstract The modern softwares have features of modularity and extensibility, and there are several researches on extensible programming languages and compilers. In this paper, we introduce Argos programming language, which provides the extensibility with the concept of plugin languages. A plugin language is used to define a method of a class, and the plugin language processors can be added and replaced dynamically. The plugin languages may be used to support multiparadigm programming or domain specific languages.

Key words : plugin, extensible language, multiparadigm, domain specific language

1. 서 론

최근 소프트웨어의 두드러진 경향은 효율적인 개발과 기능의 확장을 위해 동적으로 추가 혹은 교체할 수 있는 모듈 혹은 플러그인을 지원하는 것이다. 이러한 동적인 확장은 소프트웨어의 복잡도를 줄여주고, 각 모듈을 독립적으로 개발할 수 있도록 하기 때문에 효과적인 소프트웨어 개발을 도와준다[1]. 이러한 장점 때문에 동적인 확장성(extensibility)은 점차 많은 소프트웨어에 적용되고 있다. 예를 들어, 운영체제[2], 웹 브라우저[3], 편집기[4] 등은 기존 프로그램을 변경하지 않고, 동적인 모듈 혹은 플러그인을 이용해서 새로운 기능을 제공하는 대표적인 소프트웨어들이다.

프로그래밍 언어와 컴파일러는 사용자가 문제를 해결할 수 있도록 도와주는 소프트웨어이기 때문에 다른 소프트웨어와 마찬가지로 새로운 기능을 제공하기 위해 언어를 확장할 필요성이 있다[5]. 그러나 지금까지 언어를 확장하기 위한 연구는 주로 정적인 형태로 이루어졌다. 예를 들어, 기존 객체지향 언어에 AOP(Aspect Oriented Programming) 기능을 추가하기 위해 만들어진 AspectJ[6], 멀티패러다임 특성을 추가하기 위한 J/mp[7] 등의 언어는 문법이 고정되어 있으며, 컴파일러가 구현된 이후에는 새로운 기능을 추가할 수 없다. 반면에 매크로를 이용한 새로운 syntactic sugar를 추가하기 위한 연구[8,9]는 동적인 확장을 제공하지만, syntactic sugar는 기존 기능을 보다 편리하게 사용하도록 만들어진 것이기 때문에 새로운 기능을 제공한다고 보기는 어렵다[10].

프로그래밍 언어에서 동적인 확장성을 지원하는 연구는 상대적으로 미약한데 비해서 동적으로 확장될 수 있는 프로그램 혹은 문서의 형태는 점차 많아지고 있다. 가장 대표적인 것은 웹 페이지와 XML 문서이다. 웹 페이지는 HTML, CSS, 자바 스크립트 등의 여러 개의 언어로 구성된 프로그램으로 볼 수 있다. 웹 페이지는 또한 MathML 혹은 VRML 등과 같은 다른 언어로 작성된 프로그램 코드가 필요에 따라 포함될 수도 있다. 즉, HTML은 다른 언어를 수용할 수 있는 동적인 확장성을 지원하고 있다. 유사하게 XML 문서도 동적인 확장성을 지원한다. XML 문서는 네임스페이스를 이용해

· 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 비 회 원 : 목포대학교 정보공학부 교수
jmchoi@mokpo.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
cwyoo@ssu.ac.kr
논문접수 : 2003년 8월 4일
심사완료 : 2004년 2월 5일

서 필요에 따라 다른 XML 문서들과 결합되어 사용될 수 있다. XML은 점차 산업 전 분야로 사용이 확대되고 있으며, 여러 종류의 XML 문서들이 결합되는 경우도 많아지고 있다. 이렇게 여러 개의 언어로 작성되고, 또한 동적으로 확장될 수 있는 문서 형태는 범용 프로그래밍 언어에도 점차 영향을 미치게 될 것이다. 단적인 예로 특정 문제 영역에서 사용되는 도메인 특정 언어(DSL, Domain Specific Language)의 사용이 있다. 도메인 공학에 대한 연구가 많아짐에 따라 점차 다양한 형태의 도메인 언어들[11,12]이 개발되고 있으며, 이 언어들은 독립적으로 사용되기 어렵기 때문에 범용 언어와 API 혹은 내장된 언어 형태로 결합되어서 사용된다. 이때 다양한 도메인에 걸쳐진 문제를 해결하기 위해서는 여러 개의 도메인 언어들이 필요하고, 이 언어들이 범용 언어에 내장된 형태로 사용되기 위해서는 범용 언어가 내장 언어를 동적으로 추가할 수 있는 확장성을 가져야 한다.

프로그래밍 언어에서 동적인 확장성을 제공해야 하는 필요성을 만족시키기 위해서 본 논문에서는 프로그래밍 언어에서 모듈 단위로 동적으로 다른 언어를 추가할 수 있는 기능을 갖는 Argos 언어와 컴파일러를 소개한다. Argos 언어는 기본 언어로 자바 언어의 문법을 사용하고, 클래스의 메소드는 다른 언어를 사용할 수 있는 문법적인 확장점을 갖는다. 즉, Argos는 오퍼레이션을 여러 개의 언어와 프로그래밍 패러다임을 이용해서 확장할 수 있는 기능을 제공한다. 따라서 Argos는 기본적으로 객체지향 프로그래밍을 지원하면서, 필요한 경우에 클래스의 메소드를 C, Prolog 등과 같은 다른 프로그래밍 언어로 작성할 수 있다. Argos에서 메소드를 정의하기 위해서 사용되는 다른 프로그래밍 언어를 플러그인 언어라고 하며, 언어 개발자는 필요한 경우에 새로운 플러그인 언어와 언어 처리기를 개발해서 Argos에 동적으로 추가할 수 있다. 따라서 새로운 기능을 추가하더라도, 기존 Argos 컴파일러 혹은 Argos 언어로 작성된 소프트웨어는 변경할 필요가 없다.

현재 Argos 1.0 버전은 자바, C, Python, Prolog, SQL, XML 등의 언어를 사용할 수 있도록 지원한다. Argos 언어의 컴파일러는 DCO(delegating compiler object) 모델[13,14]에 따라 구현되었다. 즉, Argos 컴파일러는 프로그램을 사용된 플러그인 언어에 따라 코드를 분할하고, 분할된 코드를 해당 플러그인 언어 처리기에 전달한다. 처리기는 전달받은 코드를 컴파일하고, 생성된 코드를 Argos 컴파일러에 전달한다. 플러그인 언어 처리기는 플러그인 언어를 자바 코드로 변환, 해석(interpret), 혹은 실행 시에 라이브러리 레벨에서 결합될 수 있는 코드로 변환한다.

Argos와 같이 플러그인 언어를 통해 확장 가능한 언어는 멀티패러다임 프로그래밍, 도메인 특정 언어 등을 지원하기 위해서 사용될 수 있다. 여러 개의 언어로 작성된 프로그램은 기본적으로 세 가지 장점을 가지고 있다. 첫째는 문제 영역에 가장 적합한 프로그래밍 패러다임과 언어를 선택할 수 있다는 점이다. 예를 들어, 상호대화적인 기능을 필요로 하는 경우에 인터프리터 형태로 실행될 수 있는 Python과 같은 언어를 사용할 수 있고, 추론이 필요한 경우에는 Prolog와 같은 논리형 패러다임의 언어를 사용함으로써 문제를 쉽게 해결할 수 있다. 또한 도메인 언어를 플러그인 언어로 개발하는 경우에 문제 영역에 따라 적절한 도메인 언어들을 사용함으로써 생산성을 높일 수 있다. 둘째는 기존에 작성된 라이브러리를 재사용할 수 있다는 점이다. 예를 들어, C 언어는 널리 사용되기 때문에 상당히 많은 시스템들이 구축되어 있다. Argos에서 C 플러그인 언어를 사용하는 경우에 기존의 C 언어로 작성된 C 라이브러리들을 재사용할 수 있다. 셋째는 동일한 조건인 경우에 개발자의 취향에 맞는 언어를 선택해서 사용할 수 있다는 점이다. 따라서 개발자는 자신에게 익숙한 언어를 사용함으로써 보다 높은 생산성을 얻을 수 있다.

본 논문은 2장에서 언어를 확장하기 위한 기존 연구들을 소개하고, 3장에서 Argos 언어와 컴파일러에 대해서 소개한다. 4장에서는 플러그인 언어를 이용해서 구현된 응용프로그램 예를 소개하고, 5장에서는 결론 및 향후 연구 과제를 밝힌다.

2. 관련 연구

현재 프로그래밍 언어를 확장하기 위한 연구는 크게 네 가지 방향 - 새로운 프로그래밍 개념을 추가하기 위한 연구, 다른 프로그래밍 패러다임의 특성을 추가하기 위한 연구, 내장된 도메인 언어를 지원하기 위한 연구, 문법을 확장하기 위한 연구 - 으로 분류할 수 있다. 첫째로 새로운 개념을 추가하기 위한 연구는 프로그래밍 언어를 확장해서 Aspect 혹은 Multimethod 등과 같은 새로운 프로그래밍 개념을 추가하기 위한 연구이다. 이러한 연구의 예로는 AspectJ[6], MultiJava[15], RMJ (Relaxed MultiJava)[16] 등이 있다. AspectJ는 기존의 자바 언어에 AOP 기능을 추가하기 위해서 개발되었으며, 기존 자바 언어에 Aspect라는 개념을 추가한 것이다. MultiJava와 RMJ는 자바에 open class와 multi-method 개념을 추가한 것이다. 그러나 AspectJ와 MultiJava, RMJ는 모두 정적인 언어 확장만 가능하고, 동적인 언어 확장은 불가능하기 때문에 Argos와는 차별성을 갖는다.

두 번째 형태는 기존 언어에 다른 프로그래밍 패러다

임 특성을 추가하기 위한 연구이다. 이러한 연구의 대표적인 것으로는 Pizza[17], J/mp[7] 등이 있다. Pizza는 기존의 자바 언어에 함수형 패러다임의 파라메트릭 폴리모피즘(parametric polymorphism)과 STL을 지원하기 위해 확장한 언어이고, J/mp는 자바 언어에 멀티패러다임 특성을 지원하기 위해 확장한 언어이다. Pizza와 J/mp는 모두 전처리기를 이용해서 객체지향 프로그래밍 패러다임의 한계를 극복하기 위해서 사용되고 있지만, 언어 자체는 정적이며, 동적인 확장성은 제공할 수 없다.

세 번째는 기존 언어에 내장된 새로운 도메인 특정 언어를 추가하기 위한 연구이다. 이러한 연구는 주로 함수형 언어에서 이루어졌지만 점차 객체지향 언어에서도 이러한 기능을 추가하기 위한 연구가 많아지고 있다. 대표적인 연구로는 JaCo[5], JTS(Jakarta Tool Suite)[18] 등이 있다. 내장된 도메인 언어를 지원하는 경우에는 내장된 언어를 통해 언어가 동적으로 확장될 수 있다는 장점을 가지고 있다. 그러나 동적인 확장을 통해 개발된 내장된 언어는 Argos의 플러그인 언어에 비해 기능이 제한되어 있으며, 기존 언어의 라이브러리를 재사용할 수 없다는 단점을 가지고 있다.

네 번째 형태는 기존의 언어에 새로운 문법적인 기능을 추가하기 위한 연구이다. 이러한 형태의 연구는 주로 새로운 형태의 문법을 제공하거나, 자바에 매크로 기능을 추가하기 위한 목적으로 연구가 수행되고 있다. 이러한 연구로 대표적인 것은 Maya[19], JSE(Java Syntactic Extender)[9] 등이 있다. Maya와 JSE는 전처리기를 이용해서 문법적인 확장을 제공하기 때문에 문법적인 syntactic sugar을 동적으로 확장할 수 있는 방법을 제공할 수 있다. 그러나 syntactic sugar를 이용해서는 새로운 기능을 추가할 수 없으며, 다른 언어로 작성된 라이브러리를 재사용할 수도 없다.

세 번째 방법을 제외한 다른 언어 확장은 정적인데 비해 Argos는 동적인 언어 확장이 가능하다는 점이 기존 연구와 차이점이다. 내장된 도메인 언어를 사용하는 경우에도 새로운 문법과 패러다임을 갖는 언어로 확장할 수 있다는 장점은 있지만, 이것은 플러그인 언어처럼 강력한 표현력을 갖지는 못하며, 기존 언어로 구축된 라이브러리는 재사용할 수 없다는 단점을 가지고 있다. 이에 비해 Argos는 플러그인 언어에서 기존 언어로 작성된 라이브러리들을 재사용할 수 있는 장점을 가지고 있다.

언어를 확장하는 방법 이외에 기존 언어 혹은 도메인 언어를 API를 통해 결합해서 소프트웨어를 구축하는 것은 예전부터 널리 사용되던 방법이다. 그러나 API를 이용한 언어간의 결합은 Argos처럼 플러그인 언어를 사용하는 것에 비해 몇 가지 단점을 가지고 있다. 첫째는 API를 사용하는 것은 언어를 사용하는 것에 비해 표현력이 떨어진다는 점이다. API는 고정되어 있는 것에 비해 언어는 필요에 따라서 다양한 형태로 표현할 수 있기 때문이다. 둘째로 API를 사용하는 경우에는 에러가 존재하는 경우에 컴파일 시에 에러를 찾을 수 없다. 반면에 플러그인 언어를 사용하는 경우에 컴파일 시에 언어의 문법적인 에러는 물론 타입에 따른 에러들도 파악할 수 있다. 셋째는 API는 표준화가 되어 있지 않다는 것이다. 따라서 특정 API를 제공하는 라이브러리를 이용해서 개발된 소프트웨어는 다른 API를 제공하는 라이브러리를 이용하도록 변경하는 것이 상당히 어렵다. 이에 반해 플러그인 언어를 이용해서 구현하는 경우에는 문법과 의미 규칙만 만족시킨다면, 별다른 노력 없이 플러그인 언어 처리기를 다른 것으로 교체할 수 있다. 넷째로 프로그램 작성에서 API를 이용하는 것은 상당히 어렵다는 것이다. 일반적으로 API는 복잡한 함수 이름과 매개변수, 리턴 값에 대한 상당히 전문적인 지식을 가지고 있어야 한다. 이에 반해 플러그인 언어를 사용하는 경우에는 언어 자체에 대한 내용만 인지하고 있으면 된다.

3. Argos 언어와 Argos 컴파일러

3.1 Argos 언어

Argos 응용프로그램은 여러 개의 프로그래밍 언어들로 구성되어 있고, 동적으로 다른 언어를 추가할 수 있기 때문에 사용자 관점에서 Argos 프로그램은 여러 형태의 내장된 객체들로 구성된 복합 문서(compound document)로 볼 수 있다. 문서란 사용자가 화면에 보여줄 수 있고, 편집할 수 있고, 저장할 수 있는 데이타들을 묶어 놓은 개체[20]를 의미하고, 복합 문서는 문서의 개념을 확장해서 한 모델에만 소속된 것이 아닌 임의의 객체를 포함할 수 있는 문서이다[21]. 복합 문서에서 문서에 포함되는 객체는 파트(part)라고 하고, 문서 자체는 컨테이너(container)라고 한다. Argos에서 Argos 언어는 컨테이너에 해당되고, 플러그인 언어는 파트에 해당된다.

그림 1은 Argos 응용프로그램의 일반적인 형태이다. Argos 프로그램에서 플러그인 언어로 작성된 메소드는 사용된 플러그인 언어를 구별하기 위한 식별자를 함수의 헤더에 기술한다. 식별자를 기술하는 것은 "<!ns"로 시작하고, Argos에서 지원하는 모든 플러그인 언어의 식별자는 Argos 컴파일러의 구성 파일에서 관리한다.

Argos는 기본적으로 자바 언어의 문법을 따르지만, 플러그인 언어를 통해서 다른 프로그래밍 언어를 사용할 수 있다. 따라서 Argos는 플러그인 언어를 지원하기 위해서 클래스의 메소드에 다른 언어를 사용할 수 있는 확장점을 제공한다. 표 1은 자바 언어 문법[22]에 확장점을 지원하기 위해서 Argos에서 변경된 부분이다.

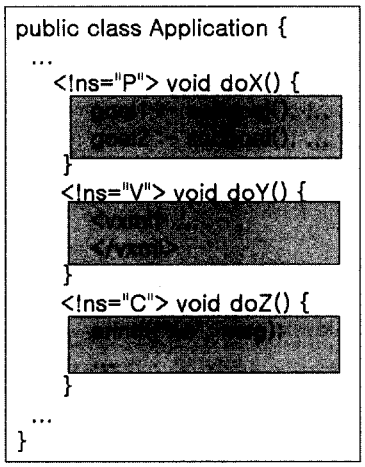

그림 1 Argos의 프로그램 형태

표 1에서 밑줄 친 부분이 Argos 언어에서 추가된 내용이다. 메소드 헤더 부분에서 플러그인 언어를 사용하기 위해서 MethodHeaderNS를 LHS(Left Hand Side)로 갖는 생성 규칙을 사용한다. MethodHeaderNS는 네임스페이스를 기술하기 위한 NameSpace를 갖고, NameSpace는 메소드에서 사용되는 언어의 유일한 ID를 URI 형식으로 기술한다. NameSpace에서 플러그인 언어 처리기에 전달하기 위한 컨텍스트에 관련된 정보들은 Context 생성 규칙의 내용을 통해서 전달된다. 플러그인 언어로 작성된 메소드의 내용을 표현하기 위해서는 ForeignMethodBody라는 생성 규칙을 사용한다. ForeignMethodBody에는 임의의 다른 프로그래밍 언어의 소스가 존재할 수 있지만, Text는 한 가지 제약 사항을 갖는다. 이것은 Text는 임의의 텍스트가 올 수 있지만, '{'와 '}' 문자가 나타나는 경우에는 짝이 맞아야 한다는 것이다. 그렇지 않은 경우에는 플러그인 언어의 내용이 어디에서 끝나는지 알 수 없기 때문에 Argos 프로그램 파싱에서 에러가 발생할 수 있다.

3.2 플러그인 언어들의 협력

Argos에서 플러그인 언어들은 서로 협력하면서 문제를 해결한다. 즉, 플러그인 언어로 작성된 클래스 메소드는 다른 일반 자바 메소드와 동일한 방법으로 호출되어 사용될 수 있다. 이때 메소드 호출을 통해서 플러그인 언어들 사이에 제어 흐름과 데이타 흐름이 발생한다. 제어 흐름은 메소드 호출을 통해서 이루어지고, 데이타 흐름은 메소드 호출 시의 매개 변수와 리턴 값의 전달과 메소드 내부에서 이루어지는 멤버 필드의 접근을 통해서 발생한다. 플러그인 언어로 작성된 메소드에서 멤버 필드를 접근하기 위해서 $ 문자를 멤버 필드 이름 앞에 붙여서 사용하고, 매개 변수를 접근하기 위해서는 $$ 문자를 접두어로 사용한다.

각 플러그인 언어는 자신만의 고유한 자료형들을 사용하기 때문에 데이타 흐름이 발생할 때는 자료형을 서로 호환될 수 있도록 변환해야 한다. 이때 한 플러그인 언어의 자료형을 다른 플러그인의 자료형으로 변환하는 방법을 사용한다면 너무 많은 자료형 변환 루틴을 필요

표 1 Argos 언어의 문법

```
<MethodDeclaration> ::= <MethodHeader> <MethodBody>
                      | <MethodHeaderNS> <ForeignMethodBody>
<MethodHeader> ::= <Modifiers>? <Type> <MethodDeclarator> <Throws>?
                      | <Modifiers>? 'void' <MethodDeclarator> <Throws>?
<MethodHeaderNS> ::= <Modifiers>? <NameSpace>
                            <Type> <MethodDeclarator> <Throws>?
<NameSpace> ::= '<!ns=' <Uri> <Context> '>'
<Uri> ::= <Q> <UriValue> <Q>
<UriValue> ::= <Identifier>
                | <UriValue> (<Alphabet> | <Digit> | ':' | '.' | '/' |'~')
<Context> ::= 'context=' <Q> <Identifier> <Q>
                | ε
<Q> ::= '₩" | '"'
<MethodDeclarator> ::= <Id> '(' <FormalParmList>? ')'
                      | <MethodDeclarator> '[' ']'
...
<MethodBody> ::= ';'
                      | <Block>
<ForeignMethodBody> ::= '{' <Text> '}'
...
```

로 한다. 예를 들어, N개의 플러그인 언어가 사용된다면, N x (N-1)개의 변환 루틴이 필요하게 된다. 이러한 문제를 해결하기 위해서 Argos는 버스 방식을 통해서 데이타 흐름이 이루어지도록 한다. 즉, 모든 플러그인 언어에서 사용되는 자료형들은 외부로 값을 전달해야 하는 경우에는 반드시 자바 자료형으로 변환하고, 역으로 외부에서 전달되는 자바 자료형은 플러그인 언어의 자료형으로 변환된다. Argos에서 사용되는 버스는 플러그인 언어들 사이의 상호 작용을 지원하기 때문에 언어 버스(language bus)라고 부른다. 언어 버스를 사용하는 경우에는 N개의 플러그인 언어를 사용하는 경우에 N개의 자료형 변환 루틴만 필요로 한다. 그림 2는 Argos 언어에서 버스 구조이다.

Argos 클래스의 멤버 필드와 매개 변수는 할당문의

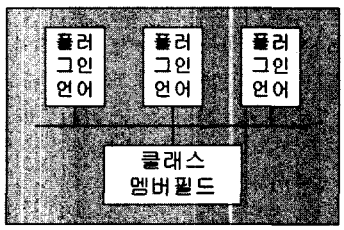

그림 2 Argos의 언어 버스

언어 버스에서 플러그인 언어로 들어오는 데이타를 임포트 데이타(import data)라고 한다. 임포트 데이타는 플러그인 언어 처리기에 의해서 자바 언어에서 플러그인 언어의 자료형으로 변환된다. 플러그인 언어에 임포트되는 데이타는 매개 변수, Argos 클래스의 멤버 필드, Argos 클래스의 메소드 호출 결과 값이 될 수 있다. 표 2는 플러그인 언어에서 가능한 임포트 데이타의 형태이다.

플러그인 언어 처리기에서 언어 버스를 통해서 외부로 전달되는 데이타를 익스포트 데이타(export data)라고 한다. 익스포트 데이타는 플러그인 언어 자료형에서 자바 자료형으로 변환되어서 전달된다. 익스포트 데이타의 예로는 Argos 클래스의 멤버 필드, 플러그인 언어의 리턴 문장, Argos 클래스의 메소드 호출에 사용되는 실매개 변수가 있다. 표 3은 플러그인 언어에서 가능한 익스포트 데이타의 형태이다.

표 2 임포트 데이타

임포트 데이타 종류	플러그인 언어에서 사용되는 형태
Argos 클래스 멤버 필드(RHS)	$name
형식 매개 변수(RHS)	$$name
Argos 클래스 메소드 호출의 결과 값	$name()

표 3 익스포트 데이타

익스포트 데이타 종류	플러그인 언어에서 사용되는 형태
Argos 클래스 멤버 필드(LHS)	$name = expression
형식 매개 변수(LHS)	$$name = expression
리턴 문장	return expression ;
실 매개 변수	$name(arg1, arg2, ...);

왼쪽(LHS, Left Hand Side)에서 사용되는 경우와 오른쪽(RHS, Right Hand Side)에서 사용되는 경우에 따라 익스포트 데이타인지 혹은 임포트 데이타인지가 결정된다. 멤버 필드나 매개 변수가 오른쪽에서 사용되는 경우에는 자바 데이타 타입을 플러그인 언어의 데이타 타입으로 변환하는 작업이 필요해진다. 반면에 왼쪽에 존재하는 경우에는 플러그인 언어의 데이타 타입을 자바 데이타 타입으로 변환해야 한다.

Argos에서 플러그인 언어들 사이의 데이타 교환은 항상 call-by-value 방식을 통해서만 이루어진다. 플러그인 언어 처리기는 자바 플랫폼에서 언어가 처리되도록 작성될 수도 있고, 하드웨어 플랫폼에서 실행되도록 작성될 수도 있다. 자바 플랫폼에서 작동되는 플러그인 언어 처리기를 사용하는 경우에는 비교적 쉽게 데이타 전달과 자료형 변환에 관련된 작업들을 처리할 수 있다. 그러나 하드웨어 플랫폼에서 실행되는 플러그인 언어 처리기를 작성하는 경우에는 자바에서 지원하는 C 기반의 JNI(Java Native Interface)[23] 인터페이스를 기반으로 데이타들이 전달된다. 따라서 X라는 언어를 위한 플러그인 처리기는 X 언어의 자료형을 C 언어 자료형으로 변환하고, 이것을 다시 JNI를 이용해서 자바와 결합하기 위한 코드들을 작성해야 한다. 그림 3은 자바 플랫폼과 하드웨어 플랫폼에 걸쳐져 있는 언어 버스의 형태를 보여준다.

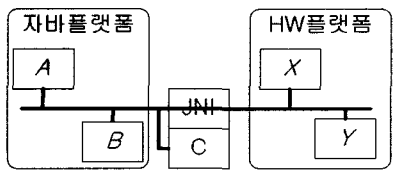

그림 3 플랫폼에 걸쳐진 언어 버스

3.3 Argos 컴파일러 구현

일반 프로그래밍 언어의 컴파일러는 어휘 분석, 구문 분석, 의미 분석을 수행하고, 최종적으로 목적 코드를 생성한다. 이때 어휘 분석과 구문 분석은 입력 프로그램이 주어진 언어의 규칙에 맞게 작성되었는지 여부를 체크하고, 규칙에 어긋난 입력인 경우에는 에러 메시지를

출력한다. 이러한 컴파일러 구조는 토큰과 문법이 고정된 언어에는 적합하지만, Argos와 같이 여러 개의 언어로 구성되어 있으며, 동적으로 확장된 언어인 경우에는 적합하지 않다. 따라서 Argos 컴파일러를 구현할 때는 2가지 요구사항을 만족시켜야 한다. 첫째는 플러그인 언어를 처리할 수 있어야 한다는 점이고, 둘째는 Argos 컴파일러가 동적으로 기능을 확장시킬 수 있어야 한다는 점이다.

하나의 소프트웨어 모듈로써 여러 개의 플러그인 언어로 구성된 Argos 프로그램을 파싱한다는 것은 상당히 복잡한 작업이다. 따라서 첫 번째 요구사항은 하나의 소프트웨어 모듈을 사용하는 대신에 각 플러그인 언어를 별도의 컴파일러를 이용해서 컴파일하고, 각 컴파일 결과를 결합하는 DCO(Delegating Compiler Object) 방법을 사용함으로써 해결할 수 있다. DCO 모델을 따르는 경우에 Argos 응용프로그램에서 각 플러그인 언어로 작성된 부분들은 Argos 컴파일러에 의해서 분리되고, 사용된 언어에 따라 각 플러그인 언어 처리기의 입력으로 전달된다. 플러그인 언어 처리기는 Argos 컴파일러를 대신해서 플러그인 언어로 작성된 프로그램을 파싱하고, 코드 생성 단계에서 생성된 결과 코드를 Argos 컴파일러에 전달한다. 따라서 Argos 컴파일러는 플러그인 언어 처리기들로 구성되어 있고, 새로운 플러그인 언어 처리기를 추가하는 경우에 Argos 응용프로그램에서 새로운 플러그인 언어도 사용할 수 있다. 그림 4는 DCO 모델에 따라 각 플러그인 언어로 작성된 코드들이 각 플러그인 언어 처리기에 의해서 파싱되는 것을 보여준다. Argos 컴파일러는 파싱 과정에서 플러그인 언어로 작성된 프로그램은 해당 플러그인 언어 처리기에 전달한다. 그림에서 점선 화살표는 플러그인 언어로 작성된 프로그램이 플러그인 언어 처리기로 전달되는 것을 보여준다.

Argos 컴파일러의 두 번째 요구사항은 동적으로 플러그인 언어를 추가할 수 있어야 한다는 점이다. 이것은 컴파일러도 다른 소프트웨어와 마찬가지로 플러그인 기능을 이용해서 동적으로 기능을 확장시킨다는 의미이다.

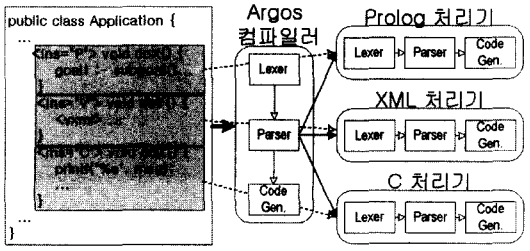

그림 4 DCO 모델을 따르는 플러그인 언어 파싱

Argos 언어에서 동적으로 추가되는 단위는 플러그인 언어이기 때문에 Argos 컴파일러에서 동적으로 기능이 추가될 수 있는 단위는 플러그인 언어 처리기가 된다. 이러한 관계에 의해서 플러그인 언어 처리기는 IPlugin이라는 인터페이스를 구현해서 작성하고, Argos 컴파일러는 구성 파일에서 등록된 플러그인 언어의 ID와 언어 처리기 클래스에 대한 정보를 읽음으로써 동적으로 해당 플러그인 언어 처리기를 생성하고, 사용할 수 있다. 그림 5는 플러그인 언어 처리기의 클래스 관계를 보여준다.

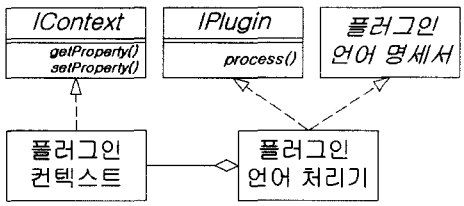

그림 5 플러그인 언어 처리기 클래스도

Argos 컴파일러에서 플러그인 언어의 ID와 플러그인 언어 처리기에 관련된 정보는 argos.conf라는 구성 파일에 관리한다. 이 파일에는 현재 Argos 언어에서 사용되는 플러그인 언어들의 URI와 플러그인 언어를 처리하기 위한 플러그인 언어 처리기 클래스들이 지정되어 있다. 표 4는 argos.conf 파일의 예를 보여준다. 표에서 tag라는 플러그인 언어를 처리하기 위해서는 argos.tag 패키지의 TagPlugin이라는 클래스를 이용한다는 것을 보여준다. Argos 컴파일러는 argos.conf 파일을 읽고서 Argos 프로그램에서 사용된 플러그인 언어를 처리하는 언어 처리기를 동적으로 로딩해서 사용할 수 있다.

표 4 argos.conf 구성 파일

#플러그인 언어 URI	플러그인 언어 처리기 클래스
tag	argos.tag.TagPlugin
sql	argos.sql.SQLPlugin
jython	argos.jython.JythonPlugin
...	

3.4 플러그인 언어 처리기

플러그인 언어 처리기는 플러그인 언어를 파싱하고, 목적 코드를 생성한다. 목적 코드는 실행 가능한 JVM(Java Virtual Machine) 코드가 될 수도 있고, 아니면 다른 고차원 언어일 수도 있다. Argos에서 사용된 플러그인 언어 처리기는 구현의 편의상 직접 JVM 코드를 생성하지 않고, 다른 고차원의 언어로 변환하거나 혹은 기존 라이브러리의 API를 이용해서 자바와 결합하는

방법을 통해서 구현된다. 이것은 전처리기 혹은 API를 사용해서 목적 코드를 생성하는 것이 처음부터 컴파일러 전체를 구축하는 것에 비해 효과적이기 때문이다. 즉, 전처리기 혹은 API를 이용하는 것은 확장된 언어를 구현하기 위한 여러 가지 방법들 중의 하나이다. 이러한 방법을 사용하면, 자바와 같이 지속적으로 발전하는 언어와 호환성을 유지하면서, 언어의 확장성은 유지할 수 있다. 이러한 이유 때문에 AspectJ, MultiJava, Pizza 등의 대부분의 확장된 언어는 전처리기를 이용해서 구현되었다.

플러그인 언어 처리기를 구현할 때는 3가지 고려할 사항이 있다. 첫째는 자료형 체크와 관련된 내용이고, 둘째는 자료형 변환, 셋째는 파싱에 관련된 문제이다. 첫째로 각 플러그인 언어들은 타입 체크 기능을 가지고 있어야 한다. 플러그인 언어에서는 매개 변수로 전달된 값과 클래스의 멤버 필드를 자유롭게 사용할 수 있다. 따라서 멤버 필드로 선언된 변수를 플러그인 언어에서 올바르게 사용하기 위해서는 항상 타입 체크를 수행해야 한다. 예를 들어, 정수형으로 선언된 멤버 필드를 플러그인 언어에서 문자열 형태로 값을 변경하는 경우에는 컴파일 에러가 발생되어야 한다. 이러한 문제점을 해결하기 위해서 Argos 컴파일러는 멤버 필드와 메소드에 관련된 정보들을 심볼 테이블을 통해 관리하고, 심볼 테이블 정보는 각 플러그인 언어 처리기에서 사용할 수 있어야 한다. 또한 Argos 응용프로그램은 기본적으로 JVM(Java Virtual Machine)에서 실행되기 때문에 실행 시에도 JVM에서 기본적인 타입 체크가 수행된다.

둘째로 자바와 플러그인 언어의 자료형들 사이에는 표준화된 매핑 방법을 사용해야 한다. 따라서 플러그인 언어와 자바 사이의 자료형 변환은 각 언어의 표준화된 명세를 이용하거나 혹은 JNI(Java Native Interface)로 표현된 자료형 매칭을 이용한다. 예를 들어, SQL 플러그인 언어는 JDBC 명세[24]에 따른 자료형 매칭을 사용하고, C 혹은 C++ 자료형과 매칭을 JNI 명세에 따른다.

셋째로 플러그인 언어 파싱에 관련되어서는 플러그인 언어를 다른 고차원 언어로 변환하고, 임포트 데이타와 익스포트 데이타를 적절하게 자료형 변환을 수행해야 하기 때문에 플러그인 언어의 문법에 따라 파싱해야 한다. 예를 들어, C 플러그인 언어 처리기는 C 플러그인 언어를 JNI 명세에 맞는 C 코드로 변환하는 역할을 한다. 이것은 비교적 간단한 작업인 듯 보이지만, 실제로 플러그인 언어 처리기가 올바르게 작동하기 위해서는 C 언어 전체를 파싱하고, AST(Abstract Syntax Tree)에서 임포트 데이타와 익스포트 데이타에 해당되는 노드를 JNI 명세에 맞는 함수 호출 노드로 변환해야 한다.

그림 6은 Argos 언어로 작성된 응용프로그램이 컴파일되고, 목적 코드를 생성하는 과정을 전체적으로 보여준다. 그림에서 사각형은 프로그램을 처리하는 처리기를 의미하고, 회색으로 채워진 사각형은 기존에 만들어진 소프트웨어를 의미한다. 그림에서 둥근 사각형은 처리기로 들어가는 입력 혹은 처리기에서 생성하는 출력 값을 의미한다. 검은색으로 채워진 화살표는 처리에서 출력으로 보낸다는 것을 표시하고, 흰색으로 채워진 화살표는 처리기의 입력으로 사용된다는 표시이다. 화살표의 끝이 선으로 된 것은 처리기가 참조해서 사용한다는 의미이다.

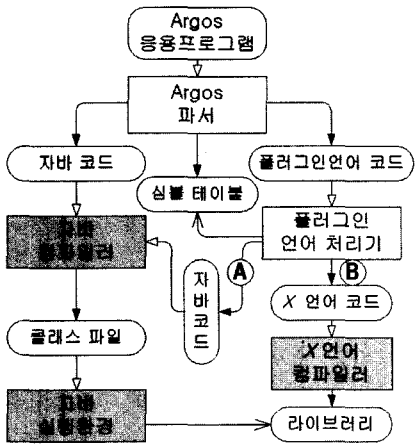

그림 6 Argos 응용프로그램 컴파일 과정

플러그인 언어는 크게 2가지 형태로 처리된다. 첫 번째는 그림 6의 A 화살표 형태로 플러그인 언어가 플러그인 언어 처리기를 통해서 자바 언어로 변환되는 경우이다. 이러한 형태의 플러그인 언어로는 SQL, Python, Prolog 등이 있다. SQL은 플러그인 언어 컴파일러를 통해서 JDBC 프로그램 코드로 변환된다. Python과 Prolog는 자바로 작성된 인터프리터가 존재하기 때문에 플러그인 언어로 작성된 메소드 내용은 처리기를 통해서 인터프리터의 입력으로 전달된다. 두 번째는 B 화살표 형태로 플러그인 언어가 자바 가상 머신이 아닌 하드웨어 플랫폼의 라이브러리로 컴파일되고, 실행 시에 결합되어서 사용되는 경우이다. 이러한 플러그인 언어의 예로는 C 언어가 있고, C 언어는 JNI를 통해서 실행 시에 자바 가상 머신과 연결된다.

Argos 컴파일러와 플러그인 언어 처리기들은 모두 LL(k) 파서 생성기인 JavaCC[25]를 이용해서 구현되었다. Argos의 문법은 J2SE 1.4를 확장한 것을 사용하고, 자바 컴파일러는 SUN의 J2SDK 1.4를 사용하였다. Python 언어의 인터프리터는 자바로 구현된 Jython[26]을 사용했으며, Prolog 인터프리터는 JavaCC를 이용해서 구현하였다. 각 플러그인 언어 처리기의 구현에 관련

된 구체적인 사항은 지면 관계상 본 논문에서는 언급하지 않는다.

4. 플러그인 언어 사용 예

Argos가 사용되는 형태를 알아보기 위해서 간단한 Argos 응용프로그램을 작성하는 경우를 예로 들어보자. 작성할 예제는 친구의 연락처 정보를 관리하는 응용프로그램이고, 사용자는 원하는 정보를 추가 혹은 검색할 수 있다. 고객은 GUI 인터페이스뿐만 아니라 기존에 구입한 VoiceXML 시스템을 이용한 음성 인터페이스를 구축하기를 원한다고 가정해보자. 이처럼 소프트웨어의 분석·설계를 수행할 때 문제 도메인과 솔루션 도메인을 동시에 고려하면서 분석·설계하는 방법을 멀티패러다임 설계라고 한다[27]. 예를 들어, 음성 인터페이스를 구축하기 위해서는 고객의 요구에 따라 반드시 VoiceXML 시스템을 사용하도록 고려하는 것을 멀티패러다임 설계라고 할 수 있다.

시스템의 요구사항은 멀티패러다임 설계에 따라 문제를 가장 잘 표현하고, 해결할 수 있는 플러그인 언어를 이용해서 해결될 수 있다. 그러나 플러그인 언어와 처리기를 작성하는 것은 초기 비용이 요구되기 때문에 플러그인 언어와 처리기는 여러 번 반복적으로 재사용될 수 있는지 여부를 고려해야 한다. 플러그인 언어가 최소 몇 번 정도 재사용될 수 있어야 그 언어를 개발하는 것이 유리한가를 결정하는 소프트웨어 공학적인 방법은 아직 연구되지 않았지만, 객체지향 프레임워크를 통한 재사용 혹은 도메인 특정 언어를 사용하는 경우와 유사할 것으로 판단된다.

개발자는 시스템이 VoiceXML 지원, 데이타 관리, 편리한 서치 기능을 만족시켜야 한다는 요구사항을 파악하고, 플러그인 언어를 사용할 것인지 여부를 결정해야 한다. 예를 들어, "VoiceXML 지원"이라는 요구사항에 대해서 XML과 HTML이 다른 응용프로그램에서도 널리 사용될 수 있기 때문에 XML을 위한 Tag라는 플러그인 언어를 구현하기로 결정할 수 있다. 또한 데이타베이스를 사용하기 위해서 SQL 플러그인 언어를 사용하기로 결정할 수 있다. 각 문제 영역에 따라 플러그인 언어와 처리기를 작성하기로 결정한 다음에는 플러그인 언어 처리기는 그림 5의 클래스 관계에 따라 IPlugin 인터페이스를 구현함으로써 독립적으로 구현될 수 있다. 작성된 플러그인 언어 처리기는 표 4의 argos.conf 구성 파일에 등록함으로써 Argos 컴파일러를 전혀 변경하지 않고, Argos 응용프로그램에서 사용될 수 있다. 만약 Argos가 동적인 확장이 아닌 정적인 방법을 이용해서 플러그인 언어의 기능을 제공한다면, Argos 언어와 컴파일러를 작성하기 위해서 상당히 많은 비용을 지불해야 했을 것이다. 즉, 새로운 Argos 컴파일러를 개발하기 위해서 개발자는 기존 Argos 컴파일러의 기능을 모두 이해해야 하고, 새로운 기능을 추가해야 하기 때문이다. 또한 정적인 확장을 사용하는 경우에 새로운 기능이 많아질수록 Argos 컴파일러의 복잡도는 더욱 증가되고, 유지보수는 더욱 힘들어지는 문제점도 발생한다.

멀티패러다임 설계를 통해서 나누어진 서브 도메인들은 객체지향 방법을 통해서 설계될 수 있다. Argos 언어를 사용하는 경우에 시스템 설계는 객체지향 방법으로 이루어지며, 클래스의 메소드들은 분석 단계에서 이루어진 서브 도메인에 맞게 작성될 수 있다. 그림 7은 멀티패러다임 설계 과정을 통해서 작성된 시스템의 클래스 관계를 보여준다[1]. 그림에서 회색 부분은 클래스와 메소드를 작성하기 위해서 사용될 플러그인 언어들을 보여준다. 사용자 인터페이스의 형태는 시각 프로그래밍과 객체지향 프로그래밍을 이용해서 작성하고, FriendManager 클래스의 메소드들은 SQL, Prolog, Tag 등의 플러그인 언어를 이용해서 작성할 수 있다.

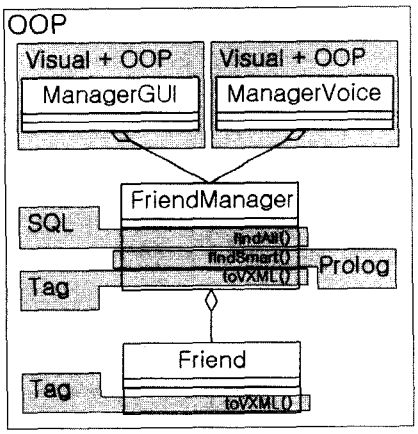

그림 7 연락처 관리 프로그램 설계

멀티패러다임 설계 과정을 통해서 만들어진 설계 내용은 Argos 언어를 사용하는 경우에 각 패러다임을 직접 매핑 방식으로 해당 플러그인 언어를 이용해서 구현할 수 있다. Friend 클래스는 이름, 전화 번호, 나이, 회사에 대한 정보들을 가지고 있다. Friend 클래스는 자신이 가지고 있는 데이타를 VoiceXML 포맷에 맞게 표현하기 위한 toVXML()이라는 메소드를 가지고 있다. toVXML() 메소드는 tag 플러그인 언어를 이용해서 데이타를 VoiceXML 문서 형태로 리턴한다. 표 5는 Argos 언어로 작성된 Friend 클래스의 코드를 보여준다.

1) 본 논문에서는 언급하지 않지만, Argos는 전용 편집기에서 시각 언어도 지원할 수 있다.

표 5 Friend 클래스

```
public class Friend {
   protected int    age;
   protected String  name; ...
   public Friend(String name, String phone, int age,
   String company) {
      this.name = name; ...
   }
   public <!ns="tag"> String toVXML() {
      <block>$name</block>
      <block>$phone</block>...
   } ...
```

표 6은 친구들에 대한 정보를 관리하기 위한 FriendManager 클래스를 Argos 언어를 이용해서 작성한 것이다. FriendManager는 데이타베이스에 접근하기 위해서는 SQL 플러그인 언어를 사용하고, 데이타베이스에서 찾아진 정보들 중에서 특별히 원하는 데이타를 찾기 위해서 Prolog 플러그인 언어를 사용할 수 있다. SQL 플러그인 언어는 프로그램에서 데이타베이스에 접근하고, 데이타를 찾아서 리턴하는 복잡한 과정들을 한 줄의 프로그램으로 처리할 수 있는 방법을 제공한다. Prolog 플러그인 언어는 주어진 데이타에서 복잡한 조건을 만족하는 데이타를 찾을 수 있는 편리한 서치 기능을 제공할 수 있다.

findAll() 메소드는 SQL 플러그인 언어를 이용해서 데이타베이스에 저장된 내용을 찾아서 리턴한다. 리턴되는 내용은 Vector 클래스에 Friend 클래스의 객체들이 저장되어 있다. findSmart() 메소드는 Prolog 플러그인 언어를 이용해서 원하는 데이타를 찾아서 리턴한다. Prolog 플러그인 언어가 수행되기 위해서 필요로 하는 지식베이스는 findSmart() 메소드의 매개 변수로 전달된다. Prolog 플러그인 언어 처리기는 지식베이스의 데이타를 이용해서 Prolog 플러그인의 질의어를 처리하고,

표 6 FriendManager 클래스

```
public class FriendManager {
   public FriendManager() {
      dbname = "friends";
      sqlc = new SQLContext(); ...
   }
    public <!ns="sql" context="sqlc"> Vector findAll() {
      [select * from $dbname]
   }
    public <!ns="prolog" context="Friend">
                        Vector   findSmart(Vector   db,
                        String company) {
      ?- friend(X, Y, Z, $$company).
   }
...
```

질의어를 만족시키는 데이타들을 Vector 타입으로 리턴한다.

표 7은 FriendManager 클래스에서 플러그인 언어들이 협동해서 작업을 수행하는 모습을 보여준다. main() 함수에서는 FriendManager 클래스의 객체를 생성하고, FriendManager를 이용해서 원하는 데이타를 찾거나, 검색할 수 있다. 이처럼 Argos를 사용하는 경우에 클래스 외부에서는 일반 자바 클래스와 Argos 클래스를 구별할 수 없다. Argos는 본 논문에서는 언급하고 있지 않지만, 매크로를 정의하고, 사용할 수 있는 기능을 가지고 있다. 표 7에서 foreach는 매크로를 통해서 정의된 syntactic sugar이다.

표 7 플러그인 언어의 협동

```
   public static void main(String args[]) {
      FriendManager fm = new FriendManager();
      Vector      data = new Vector();
      fm.add("홍길숙", "016-123-4567", 28, "삼보");
...
      data = fm.findSmart(fm.findAll(), "삼성");
      System.out.println(" << 삼성 in DB >> ");
      foreach (Friend f, data) {
         System.out.println(f.toString());
      }
...
```

5. 결 론

소프트웨어가 점차 복잡해지면서 모듈성과 확장성은 점차 중요한 개념이 되고 있다. 운영 체제, 편집기, 웹 브라우저 등의 소프트웨어에서 이미 이러한 특성을 사용하고 있으며, 점차 많은 소프트웨어에서 동적으로 추가 혹은 교체 가능한 모듈을 강조하고 있다. 이러한 추세에 따라 프로그래밍 언어와 컴파일러 분야에서도 확장성을 강조하고 있으며, 문법을 확장하기 위한 연구와 내장된 도메인 언어에 대한 연구가 점차 많아지고 있다.

본 논문에서는 플러그인 언어라는 모듈 단위로 언어를 확장할 수 있는 방법을 소개하였다. Argos는 클래스의 메소드를 플러그인 언어라는 다른 프로그래밍 언어를 이용해서 작성할 수 있도록 허용한다. Argos 컴파일러는 여러 개의 플러그인 언어로 구성된 프로그램을 컴파일하기 위해 DCO 모델에 따라 각 플러그인 언어를 별도의 플러그인 언어 처리기를 이용해서 컴파일하는 방식을 사용한다. 플러그인 언어는 문법 확장의 차원이 아닌 새로운 형태의 언어를 기존 언어에 추가하는 방식이고, 기존에 연구되지 않은 새로운 형태의 언어 확장 방법이다. 이 방법을 적용한 Argos 언어에서 플러그인 언어 처리기는 소프트웨어 플러그인 형태이기 때문에

동적으로 추가 및 교체할 수 있는 특징을 가지고 있다.

Argos 언어의 확장성은 여러 분야에서 유용하게 사용될 수 있다. 첫 번째는 멀티패러다임을 지원하기 위해서 사용될 수 있다. 큰 시스템은 하나의 언어를 이용해서 문제를 해결하기 힘들기 때문에 여러 개의 프로그래밍 패러다임을 결합해서 사용할 수 있는 방법들이 필요하고, Argos는 다른 프로그래밍 패러다임의 언어를 플러그인 언어로 사용함으로써 이러한 필요성을 효과적으로 지원할 수 있다.

두 번째로 사용될 수 있는 분야는 도메인 언어의 지원이다. 도메인 언어는 응용프로그램에서 특정 문제만을 해결하기 위해서 사용되며, 일반적으로 호스트 언어에 내장되어서 사용되는 특징을 가지고 있다. 도메인 언어를 사용하는 경우에 특정 문제를 쉽게 해결할 수 있는 장점을 가지고 있다. Argos를 사용하는 경우에 도메인 언어를 플러그인 언어 형태로 구현함으로써 도메인 언어를 사용하도록 지원할 수 있다.

셋째는 기존 언어를 플러그인 언어로 사용하는 경우에 이미 작성된 라이브러리를 재사용할 수 있다는 점이다. 예를 들어, C 플러그인 언어는 기존에 작성된 C 라이브러리를 그대로 재사용할 수 있게 지원한다.

플러그인 언어 단위로 확장할 수 있는 Argos 언어는 멀티패러다임 프로그래밍, 도메인 언어, 기존 라이브러리를 재사용해야 하는 필요성이 있는 경우에 효과적으로 사용될 수 있을 것이다.

참 고 문 헌

[1] Johannes Mayers, Ingo Melzer, and Franz Schweiggert, "Lightweight Plug-in-Based application Development," In *Objects, Components, Architectures, Services, and Applications for a Networked World*, LNCS 2591, Springer-Verlag, pp. 87-102, 2003.

[2] Alessandro Rubini, "Dynamic Kernels: Modularized Device Drivers," In *Linux Journal, Mar.*, 1996, available at http://www.linuxjournal.com/.

[3] Larry Hoff, "Netscape Plug-Ins", In *Linux Journal*, Sep., 1999, available at http://www.linuxjournal.com/.

[4] Eclipse Platform Technical Overview, Object Technology International, Inc., Feb., 2003, available at http://www.eclipse.org/.

[5] Matthias Zenger and Martin Odersky, "Implementing Extensible Compilers," In *Proc. of MPOOL*, pp. 61-80, 2001.

[6] AspectJ Project, http://eclipse.org/aspectj/.

[7] Timothy A. Budd, "The Return of Jensen's Device," In *Proc. of MPOOL*, pp. 45-63, 2002.

[8] Jason Baker and Wilson C. Hsieh, "Maya: Multiple-Dispatch Syntax Extension in Java," In *Proc. of PLDI*, pp. 270-281, 2002.

[9] Jonathan Bachrach and Keith Playford, "The Java Syntactic Extender(JSE)," In *Proc. of OOPSLA*, pp. 31-42, 2001.

[10] Free Online Dictionary of Computing, http://wombat.doc.ic.ac.uk/foldoc/.

[11] Conal Elliott, "Modeling Interactive 3D and Multimedia Animation with an Embedded Language," In *Proc. of USENIX on Domain-Specific Languages*, pp. 285-296, 1997.

[12] Scott Thibault, Renaud Marlet, and Charles Consel, "A Domain-Specific Language for Video Device Driver: from Design to Implementation," In *Proc. of USENIX on Domain-Specific Languages*, 1997.

[13] Jan Bosch, "Delegating Compiler Objects: Modularity and Reusability in Language Engineering," In *Nordic Journal of Computing*, 4, pp. 66-92, 1997.

[14] Jan Bosch, *Layered Object Model Investigating Paradigm Extensibility*, Ph.D. Thesis, Dept. of CS., Lund Univ., Sweden, 1995.

[15] Curtis Clifton, et. al., "MultiJava: Modular Open Classes and Symmetric Multiple Dispatch for Java," In *Proc. of OOPSLA*, pp. 130-145, 2000.

[16] Todd Millstein, Mark Reay, and Craig Chambers, "Relaxed MultiJava: Balancing Extensibility and Modular Typechecking," In *Proc. of OOPSLA*, pp. 224-240, 2003.

[17] Martin Odersky and Philip Wadler, "Pizza into Java: Translating theory into practice," In *Proc. 24th ACM Symposium on Principles of Programming Languages*, pp. 146-159, 1997.

[18] Don Batory, Bernie Lofaso, and Yannis Smaragdakis, "JTS: Tools for Implementing Domain-Specific Languages," In *Proc. of Software Reuse*, pp. 143-153, 1998.

[19] Jason Baker and Wilson C. Hsieh, "Maya: Multiple-Dispatch Syntax Extension in Java," In *Proc. of PLDI*, pp. 270 - 281, 2002.

[20] Wolfgang Weck, "Document-Centered Computing: Compound Document Editors as User Interfaces," In *Journal of Symbolic Computation*, no. 11, pp. 1-24, 1997.

[21] Clemens Szyperski, Dominik Gruntz, and Stephan Murer, *Component Software, 2nd ed.*, Addison-Wesley, 2002.

[22] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, *The Java Language Specification*, 2nd ed., Addison-Wesley, 2000. available at http://java.sun.com/docs/books/jls/.

[23] Sheng Liang, *Java Native Interface: Programmer's Guide and Specification*, Addison-Wesley, 1999.

[24] Seth White, et al., *JDBC API Tutorial and Reference : Universal Data Access for the Java 2 Platform, 2nd ed.*, Addison-Wesley, 1999.

[25] JavaCC-The Java Parser Generator, available at https://javacc.dev.java.net/.
[26] Jim Hugunin, "Python and Java: The Best of Both Worlds," In *Proc. of the International Python Conference*, 1997, available at http://www.jython.org/.
[27] James O. Coplien, *Multi-Paradigm Design for C++*, Addison-Wesley, 1999.

최 종 명
정보과학회 논문지 : 소프트웨어 및 응용
제 31 권 제 2 호 참조

유 재 우
정보과학회 논문지 : 소프트웨어 및 응용
제 31 권 제 1 호 참조