결합 방식 멀티패러다임 프로그래밍을 지원하는 언어의 설계 및 구현

최 종 명† · 유 재 우††

요 약

본 논문에서는 멀티패러다임 프로그래밍의 결합 방식(compositional approach)[20]을 프로그래밍 언어에 적용한 새로운 형태의 멀티패러다임 언어인 Argos를 소개한다. Argo는 자바 언어의 수퍼셋이고, Argos 언어의 메소드를 정의하는 문법은 다른 언어의 문법을 사용할 수 있는 확장점을 가지고 있다. 따라서 Argos 클래스의 각 메소드는 자바, C, Prolog, Python 등의 여러 프로그래밍 언어들 중에서 하나를 선택해서 구현할 수 있도록 허용함으로써 객체지향과 멀티패러다임 프로그래밍을 동시에 지원한다. Argos의 메소드는 기존의 프로그래밍 언어로 작성할 수 있기 때문에 다른 멀티패러다임 언어에 비해 상대적으로 배우기 쉽고, 라이브러리 재사용성도 높은 장점을 가지고 있다. Argos 컴파일러는 입력 프로그램을 사용된 언어에 따라 분할하고, 분할된 메소드 코드를 해당 언어의 처리기에 전달해서 컴파일하는 DCO(delegating compiler object) 모델[28, 29]에 따라 구현된다.

Design and Implementation of a Language Supporting Compositional Approach to Multiparadigm Programming

Jong-Myung Choi† · Chae-Woo Yoo††

ABSTRACT

In this paper, we introduce a new style multiparadigm language named Argos which applies a compositional approach [20] to multiparadigm programming. Argos is a superset of the Java, and its grammar has an extension point which allows other languages to be used in Argos programs. Therefore, Argos can support object-oriented programming and multiparadigm programming by enabling each method in a class to be implemented with one of the Java, C, Prolog, Python, and XML languages. Since Argos allows the existing languages to be used, it has advantages such as easiness of learning and high reusability. The Argos compiler is implemented according to the delegating compiler object (DCO) model[28, 29]. The compiler partitions a program into several parts according to the languages used in methods and delivers the parts the languages' processors which compile the parts.

키워드 : 멀티패러다임(Multiparadigm), 객체지향(Object-oriented), 플러그인 언어(Plugin Language)

1. 서 론

컴퓨터 시스템이 발전하면서 개발해야 하는 소프트웨어는 점차 복잡하고 많은 요구 사항들을 만족시켜야 한다. 복잡한 소프트웨어는 근본적으로 하나의 프로그래밍 패러다임 혹은 언어를 이용해서 개발하는 것 보다는 여러 개의 프로그래밍 패러다임과 언어를 이용해서 개발하는 것이 효과적이다[1, 12, 16, 17, 20, 23]. 프로그래밍 패러다임(programming paradigm)은 "프로그래머의 의도를 표현하기 위한 프로그래밍 스타일(a style of programming expressing the programmer's intent)"[22], "문제를 해결하기 위한 접근법 혹은 모델(model or approach in solving a problem)"[2]을 의미한다. 각 프로그래밍 패러다임은 문제를 해결하기 위한 패턴을 가지고 있으며, 각 패턴들은 모든 문제에 다 적합하게 적용되는 것은 아니다. 따라서 어떤 문제는 함수형 패러다임을 적용하는 것이 효과적일 수 있고, 또 다른 문제는 객체지향 패러다임을 적용하는 것이 좋은 해결책이 될 수 있다. 좋은 해결이란 것은 해법이 보다 명확하고, 읽기 쉬우며, 간단하며, 문제를 직접 모델링할 수 있다는 의미이다[1]. 간단하지 않은 소프트웨어를 설계할 때 시스템의 모든 부분들이 동일한 패러다임을 이용해서 해결되지 않기 때문에 여러 개의 프로그래밍 패러다임을 혼합해서 사용해야 한다. 이러한 프로그래밍 방법을 멀티패러다임 프로그래밍이라고 한다.

개발해야 할 소프트웨어의 각 문제 영역을 효율적으로

※ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 정 회 원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2003년 7월 4일, 심사완료 : 2003년 11월 27일

해결하기 위해서 멀티패러다임 언어에 대한 많은 연구[3, 4, 11, 15, 32-35]가 수행되었다. 그러나 지금까지 연구된 멀티패러다임 언어는 여러 가지 문제점들을 가지고 있다. 첫째로 가장 큰 문제점은 기존에 널리 사용되는 언어들로 작성된 라이브러리들을 재사용이 제한적이거나 재사용을 할 수 없다는 점이다[1]. 지금까지 연구된 멀티패러다임 언어들은 여러 개의 프로그래밍 패러다임을 결합하기 위한 방법에 초점을 기울였기 때문에 현재 널리 사용되는 언어로 작성된 라이브러리들을 재사용하는 것에는 상대적으로 소홀했다. 따라서 새로 개발된 멀티패러다임 언어는 라이브러리가 부족해서 널리 사용되기 어려웠다. 둘째는 멀티패러다임 언어가 새로운 문법을 사용하기 때문에 기존 프로그래머는 새로 언어를 배워야 하는 문제점을 가지고 있다[1, 15]. 또한 멀티패러다임 언어는 여러 개의 패러다임을 동시에 사용하기 때문에 기존 프로그래머들이 배우기도 어렵다는 문제점도 가지고 있다.

본 논문에서는 시스템 레벨에서 사용되던 결합 방식(compositional approach)의 멀티패러다임 프로그래밍 방법을 프로그래밍 언어에 적용한 새로운 형태의 멀티패러다임 언어를 소개한다. 결합 방식에서 멀티패러다임 프로그램은 "단일 패러다임 프로그램들의 모임(a collection of single-paradigm programs)"이다[20]. 즉, 멀티패러다임 프로그램은 단일 패러다임으로 작성된 여러 개의 모듈들의 결합이다. 이 방법에 따라서 논문에서는 객체지향 클래스의 각 메소드를 여러 개의 프로그래밍 언어들 중에서 하나를 선택해서 구현할 수 있는 새로운 형태의 멀티패러다임 언어인 Argos를 소개한다. 이것은 Argos 언어의 문법이 다른 언어 문법을 사용할 수 있는 확장점을 제공하기 때문에 가능한 것이다. Argos 클래스의 메소드는 기존에 널리 사용되던 언어를 이용해서 구현되기 때문에 프로그래머는 새로운 언어를 배워야 하는 부담감을 줄일 수 있고, 기존 언어에서 사용되던 라이브러리를 재사용할 수 있다는 장점을 가지고 있다. Argos 언어는 자바 언어의 수퍼셋이고, 현재 Argos는 자바, C, Python, Prolog, SQL, XML 등의 언어를 사용할 수 있도록 지원한다. Argos 언어의 컴파일은 DCO(delegating compiler object) 모델[28, 29]에 따라 이루어진다. 즉, Argos 컴파일러는 프로그램을 사용된 언어에 따라 코드를 분할하고, 분할된 코드를 해당 언어의 처리기에 전달한다. 처리기는 전달받은 코드를 컴파일하고, 생성된 코드를 Argos 컴파일러에 전달한다.

Argos는 객체지향과 멀티패러다임 프로그래밍을 자연스럽게 결합하기 때문에 객체지향 프로그래밍의 장점인 자연스러운 모델링, 재사용 등의 장점을 그대로 유지할 수 있으면서, 문제 영역에 따라 패러다임과 언어를 선택할 수 있다는 멀티패러다임 프로그래밍의 장점을 동시에 가지고 있다.

본 논문은 2장에서 관련 연구를 소개하고, 3장에서 Argos의 언어 모델을 제시하고, 4장에서는 플러그인 언어들 사이의 상호 작용 방법을 기술한다. 5장은 Argos 언어의 컴파일러의 구현에 대한 내용을 소개하고, 6장에서는 Argos 언어를 사용하는 간단한 예제 프로그램을 소개한다. 마지막으로 7장에서 결론과 향후 연구 과제를 밝힌다.

2. 관련 연구

2.1 멀티패러다임 지원 시스템

시스템 레벨에서 멀티패러다임 프로그래밍은 상대적으로 쉽게 지원된다. 시스템 레벨에서는 Pamela Zave[20]가 제안한 결합 방식에 따라서 멀티패러다임 프로그래밍이 이루어진다. 예를 들어, UNIX 운영체제는 멀티패러다임을 지원하는 것으로 인식되고 있다[14, 23]. 즉, UNIX 운영체제에서는 파이프라인을 이용해서 여러 개의 프로그래밍 언어(sed, awk 등)들이 결합되어서 문제를 해결하기 때문이다.

UNIX 이외의 보다 현대적인 시스템에서도 멀티패러다임을 지원하기 위한 노력은 계속되고 있다. 이러한 시스템의 예로는 .NET의 CLR(Common Language Runtime)[25], IBM의 SOM(System Object Model)[26], MS의 COM(Common Object Model)[27] 등이 존재한다. .NET의 CLR은 여러 개의 프로그래밍 언어를 지원하면서, 런타임 레벨에서 프로그래밍 언어들을 결합할 수 있도록 지원하고, 이러한 목표를 위해서 .NET에서는 CTS(Common Type System)과 CIL(Common Intermediate Language)을 제공한다. IBM의 SOM은 언어들이 서로 협력할 수 있는 언어에 독립적인 런타임을 제공하고, SOM 클래스의 인터페이스는 OMG CORBA의 IDL를 이용해서 정의된다. SOM IDL 컴파일러는 타겟 컴파일러를 위한 바인딩 언어와 IDL 클래스 정의에 맞는 구현 언어를 생성한다. COM은 비교적 간단한 인터페이스를 통해서 다양한 언어들이 서로 협력해서 작동할 수 있는 방법을 제공한다. COM에서 모든 메소드는 주어진 포맷의 에러 번호를 리턴해야 하고, 메소드 호출을 위한 매개 변수는 기본형만 가능하다는 제약 사항이 있다.

시스템 레벨에서 제공되는 멀티패러다임 프로그래밍은 확장성, 개방성, 사용의 용이성이라는 장점을 가지고 있다. 그러나 특정 플랫폼에서만 사용할 수 있고, 언어 레벨에서 지원되지 않기 때문에 효과적인 개발 도구나 개발 방법론에 대한 연구가 상대적으로 이루어지기 않았다는 문제점을 가지고 있다.

2.2 멀티패러다임 프로그래밍 언어

멀티패러다임 프로그래밍을 언어 수준에서 지원하기 위한 언어에 관한 연구는 많이 수행되었으며, 이러한 언어들에 관한 광범위한 조사는 Kam-Wing[11]과 Diomidis[23]에 의해 연구되었다. 멀티패러다임 언어들은 여러 가지 방법으로 구현될 수 있지만, Brent Hailpen에 따르면 언어의 구현 방법에 따라 크게 4가지 형태로 구분할 수 있다[21].

- 기존의 언어들의 문법과 의미를 결합하는 방법
- 기존 언어에 새로운 언어적인 구조를 추가하는 방법
- 새로운 이론하에 기존의 언어를 재정의하는 방법
- 완전히 새로운 언어를 작성하는 방법

첫 번째 방법은 기존 언어를 변경해서 다른 프로그래밍 패러다임의 문법 혹은 의미적 특성을 갖는 언어를 새로 작성하는 방법이다. 예를 들어, LogiC++[32], DLP[33] 등의 언어는 객체지향과 논리형 패러다임을 결합하기 위해서 C++ 클래스의 메소드를 Prolog의 절을 이용해서 기술하는 방법을 사용하였고, Curry[34]는 함수형 언어와 논리형 언어를 결합하는 언어이다. 이 방법의 특징은 베이스 언어의 특징을 기반으로 새로운 패러다임적인 특성을 부여한 새로운 언어를 작성한다는 것이다. 따라서 이 방법은 사용자가 최소한 하나의 기존 언어에 익숙하다는 장점을 가지고 있는 반면에 여러 개의 언어가 결합되기 때문에 문법과 의미가 복잡하고, 기존 라이브러리를 재사용하기 어렵다는 단점을 가지고 있다[21].

두 번째 방법은 기존의 언어를 기반으로 새로운 멀티패러다임의 특성을 위해 문법 구조를 추가하는 방법으로 C++, Pizza[31] 등의 언어에서 널리 사용되는 방법이다. 이 방법은 기존 언어로 작성된 소프트웨어를 새로운 언어에서도 사용할 수 있다는 장점을 가지고 있다. 예를 들어, C++는 C를 확장해서 객체지향 패러다임을 지원하는 멀티패러다임 언어[23]이며, C++의 바탕이 되는 C 언어의 라이브러리를 그대로 사용할 수 있다. 또한 Pizza는 자바 언어를 확장해서 함수형 패러다임의 파라메트릭 폴리머피즘을 지원하는 멀티패러다임 언어로서 기존 자바 언어의 기능을 그대로 사용할 수 있다. 그러나 이 방법도 라이브러리의 재사용이 일부분만 가능하다는 단점이 있다. 예를 들어, Pizza의 경우에 기본이 되는 자바 언어의 라이브러리만 재사용할 수 있고, 함수형 언어의 라이브러리는 재사용할 수 없다.

세 번째 방법은 기존 언어를 새로운 이론을 기반으로 불필요한 기능을 제거하고, 새로운 패러다임적인 특성을 부과해서 새로운 멀티패러다임 언어를 구현하는 방법이다[21]. 이러한 언어의 예로는 논리 언어와 함수형 언어를 결합한 LIFE[35]가 있다.

네 번째 방법은 처음부터 멀티패러다임을 위해 새로운 언어를 정의하는 방법이다. 이 방법의 대표적인 언어로는 Leda[3]와 I++[4]가 있다. Leda는 명령형, 객체지향, 함수형, 논리 프로그래밍 패러다임을 지원하기 위해서 설계되었다. I+는 객체지향, 논리형, 함수형 패러다임을 지원하는 멀티패러다임 프로그래밍 언어이다[4]. I+에서 프로그램은 클래스와 질의어들로 구성되어 있고, 클래스는 메소드의 형태에 따라 논리 클래스와 함수 클래스로 구분되어 있다. 이러한 방법으로 개발된 언어는 일관성이 있고, 우아하다는 장점을 가지고 있다[21]. 그러나 복잡하기 때문에 구현이 어렵고, 배우기 어렵고, 라이브러리 재사용이 안 된다는 단점이 있다.

다른 방법으로 구현된 언어에 비해 두 번째 방법으로 구현된 언어가 널리 사용된다. 이것은 두 번째 방법으로 구현된 언어가 기존 라이브러리, 개발 도구 등을 지원하고, 상대적으로 배우기 쉽기 때문이다. 이것은 Leda 언어를 개발한 Timothy의 연구에서도 볼 수 있다. Timothy는 Leda 이후에 자바 언어를 멀티패러다임 언어로 확장한 J/mp[15]라는 언어를 개발하였다. Leda와 J/mp의 가장 큰 차이점은 Leda는 멀티패러다임을 지원하기 위해 새로 설계된 언어인 반면에 J/mp에서는 자바 언어를 확장하는 방향으로 구현되었다. Timothy는 J/mp에서 자바 언어의 기본적은 문법과 의미를 유지면서 멀티패러다임 특성을 자연스럽게 자바 언어와 결합되도록 하는 것에 중점을 두고 있다. 이처럼 기존 언어의 문법, 의미, 라이브러리, 도구들을 지원하지 않는 새로운 언어는 실세계에서 살아남기 어렵다.

기존의 멀티패러다임 언어들은 모두 여러 개의 패러다임적인 특성들을 하나의 고정된 문법 구조를 이용해서 표현하려고 노력하였다. 그렇기 때문에 언어의 변형이 요구되고, 기존 라이브러리를 재사용하기 어렵고, 배우기 어려운 문제점을 가지고 있다. 이에 반해 Argos는 시스템 레벨에서 사용되던 결합 방식을 언어 수준에 적용하는 방법을 사용한다. 즉, Argos의 문법은 고정되어 있지만, 다른 언어의 문법을 추가할 수 있는 확장점을 가지고 있다. 따라서 Argos는 확장점을 통해서 다른 프로그래밍 언어들을 결합해서 멀티패러다임을 지원한다. 확장점을 통해서 결합되는 언어들은 기존 언어들이기 때문에 사용자가 배우기 쉽고, 기존 라이브러리들을 재사용할 수 있다. Argos는 또한 멀티패러다임 프로그래밍을 다른 언어에 비해 효과적으로 지원할 수 있다. 이것은 프로그래밍 패러다임은 해당 패러다임에 속한 언어의 문법을 사용할 때 가장 잘 표현되는데[19], Argos는 각 프로그래밍 패러다임에 소속된 언어의 문법과 의미를 그대로 지원하기 때문이다.

3. 클래스내의 멀티패러다임

3.1 멀티패러다임 프로그램 단위

대부분의 프로그래밍 언어들은 명시적이건 혹은 묵시적이건 값을 표현하기 위해서 자료형을 사용한다. 이러한 자료형들은 언어에 따라 이름은 달라질 수 있지만 기본적인 형태는 서로 유사하다. 따라서 대부분의 언어에서 사용하는 자료형들은 기본적으로 공통되는 내용들을 가지고 있다. 반면에 문제를 해결하기 위해서 사용되는 문제 해결 방법은 상당히 다르다. 예를 들어 명령형 패러다임을 사용하는 C, Pascal 등의 언어에서 문제를 해결하는 방법과 함수형 패러다임을 사용하는 Haskell 언어에서 사용되는 방법은 상당히 달라진다. 따라서 문제를 표현하고, 해결하기 위한 언어적인 특성들은 언어의 자료형도 중요하겠지만, 언어의 문제

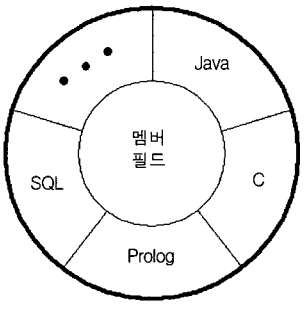
해결 방법에 따라 많이 좌우된다.

각 프로그래밍 패러다임의 언어들은 문제의 복잡성을 해결하기 위해서 추상화된 개념을 제공한다. 프로그래밍 패러다임에 따라 추상화 단위와 추상화 단위의 이름은 다르지만, 각 프로그래밍 패러다임에서 프로그래머는 추상화된 단위를 하나의 개체(entity)로 취급하고, 문제를 추상화된 개체들을 이용해서 해결한다. <표 1>은 각 프로그래밍 패러다임별로 사용되는 추상화 단위의 이름 및 형태를 보여준다.

〈표 1〉 프로그래밍 패러다임별 추상화 단위

패러다임	추상화 단위	형 태
명 령 형	함수, 프로시저	오퍼레이션
함 수 형	함수	오퍼레이션
논 리 형	절	오퍼레이션
객체지향	클래스	Σ(오퍼레이션) + Σ(상태)

객체지향 패러다임을 제외한 다른 프로그래밍 패러다임에서 추상화의 단위는 특정 문제 하나를 해결하기 위한 오퍼레이션이다. 이에 비해서 객체지향 패러다임의 추상화 단위인 클래스는 여러 개의 오퍼레이션과 상태를 가지고 있다. 클래스는 응집도가 높은 데이터와 오퍼레이션을 하나의 개념으로 표현한 것이고, 클래스 내부는 외부에서 접근할 수 없는 캡슐화 기능을 제공하기 때문에 클래스의 메소드가 어떻게 구현되었는지 여부는 사용자에게는 관심 밖의 일이다. 따라서 객체의 메소드들을 한 개의 언어를 이용해서 작성하는 대신에 "Multiparadigm within a Class"[18] 모델에 따라서 서로 다른 여러 개의 프로그래밍 언어를 이용해서 작성하는 것이 가능하다. 이러한 경우에 각 메소드들은 여러 개의 다른 프로그래밍 언어로 작성된 작은 프로그램이 된다. 또한 이 작은 프로그램들은 객체라는 작은 컨텍스트에서 작동하게 된다. 따라서 객체의 멤버 필드들은 언어의 컨텍스트에 해당된다. 이 모델에 따라 Argos 클래스의 메소드는 여러 개의 프로그래밍 언어 중에서 선택된 하나의 언어로 작성된다. 이때 사용된 언어를 플러그인 언어라고 한다. (그림 1)은 Argos 언어에서 클래스의 메소드는 다른 프로그래밍 패러다임의 플러그인 언어를 통해서 구현되고, 메소드들은 멤버 필드를 공유한다는 것을 보여준다.



(그림 1) Argos 클래스

객체의 각 메소드들을 작은 프로그램들로 보는 관점은 여러 가지 장점을 가지고 있다. 첫 번째는 객체지향 프로그래밍에서 멀티패러다임 프로그래밍을 자연스럽고 일관된 방법으로 지원할 수 있다는 것이다. 따라서 개발해야 할 소프트웨어의 문제를 객체지향과 컴포넌트 방법을 이용해서 분석, 설계할 수 있고, 클래스 내부에서 세부적인 문제 영역은 패러다임별로 분할해서 각 패러다임에 가장 적합한 언어를 선택해서 문제를 해결할 수 있다. 두 번째는 언어의 기본적인 형태는 변경하지 않고 사용할 수 있다는 점이다. 기존의 멀티패러다임 언어는 완전히 새로운 언어를 만들기 때문에 개발자가 새로운 언어를 배워야하고, 기존의 라이브러리를 재사용할 수 없는 문제점을 가지고 있었다. 그러나 Argos에서 각 메소드는 C, Python, Prolog 등과 같은 언어의 문법을 그대로 사용할 뿐만 아니라 기존의 라이브러리들도 재사용할 수 있다는 장점을 가지고 있다.

3.2 사용자 모델

Argos에서 응용프로그램은 여러 개의 프로그래밍 언어들로 구성되어 있다. 따라서 사용자 관점에서 Argos 프로그램은 여러 형태의 내장된 객체들로 구성된 문서로 볼 수 있다. 문서란 사용자가 화면에 보여줄 수 있고, 편집할 수 있고, 저장할 수 있는 데이터들을 묶어 놓은 개체를 의미한다[5]. Argos에서 플러그인 언어들은 복합 문서 모델에 따라 작성된다. 복합 문서(compound document)는 문서의 개념을 확장해서 한 모델에만 소속된 것이 아닌 임의의 객체를 포함할 수 있는 문서이다[6, 7]. 복합 문서에 포함된 객체를 문서의 파트(part) 혹은 컴포넌트라고 하고, 문서는 파트들을 포함하기 위한 컨테이너의 역할을 한다. Argos에서 자바 언어는 컨테이너가 되고, 플러그인 언어들은 문서에 내장되는 파트로 볼 수 있다. 복합 문서에서 파트는 다른 파트를 포함할 수 있도록 허용함으로써 재귀적인 포함 관계를 사용하는 경우도 있다. 그러나 현재 구현된 Argos에서는 재귀적인 포함은 허용하지 않는다.

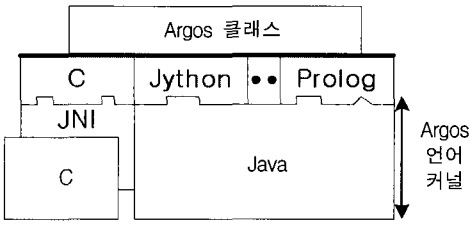
```
public class Application {
  ...
    <!ns="P"> void doX() {
        goal1 :- subgoal(), ...
        goal2 :- subgoal(), ...
    }
    <!ns="V"> void doY() {
        <vxml> ...
        </vxml>
    }
    <!ns="C"> void doZ() {
        printf("%s", msg);
        ...
    }
  ...
}
```

(그림 2) Argos의 프로그램 형태

(그림 2)는 Argos 언어에서 사용되는 복합 문서 모델의 형태이다. Argos 문서에는 컨테이너로 자바 언어를 사용하고, 다른 프로그래밍 언어들을 플러그인 언어로 문서에 내장되어 사용될 수 있다. 각 플러그인 언어들은 종류를 구별하기 위한 유일한 식별자를 갖는다.

3.3 Argos 커널 언어

Argos 언어는 Argos 커널 언어와 플러그인 언어들로 구성되고, Argos 커널 언어는 멀티패러다임을 지원하기 위해서 설계된 언어이다. Argos 커널 언어는 자바 언어에 다른 프로그래밍 언어를 추가할 수 있는 기능을 제공함으로써 자바 언어에 확장성을 부여한다. 따라서 Argos 커널 언어는 자바의 수퍼셋이기 때문에 자바 언어의 문법과 객체지향적인 특징 및 플랫폼 독립성을 그대로 지원한다. (그림 3)은 Argos 커널 언어의 구성을 보여준다.



(그림 3) Argos 커널 언어 구조

Argos 언어가 자바 언어를 기반으로 작성된 이유는 자바 언어가 여러 가지 장점들을 가지고 있기 때문이다. 첫째는 객체지향 언어에서 가장 널리 사용되는 언어라는 점이다. 자바 언어는 많은 개발자들에 의해서 사용되고 있으며, 따라서 많은 라이브러리들이 축적되어 있다. Argos는 자바 언어를 기초로 하기 때문에 자바 개발자들에게 익숙하고, 자바 라이브러리들을 그대로 재사용할 수 있는 장점을 가지고 있다. 둘째는 자바의 플랫폼에 독립적인 장점 때문이다. Argos는 순수 자바 기반의 언어이기 때문에 플랫폼 독립성이 뛰어나다. 셋째는 자바가 컴포넌트 프로그래밍을 지원할 수 있다는 점 때문이다. 컴포넌트 프로그래밍은 점차 중요해지고 있기 때문에 앞으로 널리 사용되기 위해서는 컴포넌트 프로그래밍을 지원할 수 있어야 한다.

Argos 언어는 자바 언의 수퍼셋이기 때문에 기본적으로 자바 언어의 문법을 따르지만, 클래스의 메소드 부분은 자바 언어의 문법[8]을 확장한 것이다. <표 2>는 Argos 언어에서 플러그인 언어의 확장을 지원하기 위한 문법이다.

<표 2> 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> '[' ']'
<FormalParmList> ::= <FormalParam>
        | <FormalParmList> ',' <FormalParam>
<FormalParam> ::= 'final'? <Type> <VarDeclId>
<Throws> ::= 'throws' <ClassTypeList>
<ClassTypeList> ::= <ClassType>
        | <ClassTypeList>, <ClassType>
<MethodBody> ::= ';'
        | <Block>
<ForeignMethodBody> ::= '{' <Text> '}'
...
```

<표 2>에서 밑줄 친 부분이 Argos 언어에서 추가된 내용이다. 메소드 헤더 부분에서 플러그인 언어를 사용하기 위해서 MethodHeaderNS라는 넌터미널을 사용한다. MethodHeaderNS는 네임스페이스를 기술하기 위한 NameSpace를 갖는다. NameSpace는 플러그인 언어의 유일한 ID를 URI 형식으로 기술한다. NameSpace에서 플러그인 언어 처리기에 전달하기 위한 컨텍스트에 관련된 정보들은 Context를 통해서 전달된다. 플러그인 언어로 작성된 메소드의 내용을 표현하기 위해서는 ForeignMethodBody라는 넌터미널을 사용한다. ForeignMethodBody에는 다른 프로그래밍 언어의 소스가 나타날 수 있다. Text는 한 가지 제약 사항을 갖는데, 이것은 Text는 임의의 텍스트가 올 수 있지만, '{'와 '}' 문자가 나타나는 경우에는 짝이 맞아야 한다는 점이다. 그렇지 않은 경우에는 플러그인 언어의 내용이 어디에서 끝나는지 알 수 없기 때문에 파싱에 문제가 발생한다.

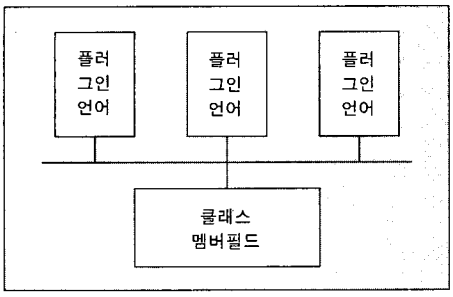
4. 멀티패러다임 언어들 간의 상호작용

4.1 Argos 언어에서 플러그인 언어들의 상호 작용 모델

Argos 프로그램에서 사용되는 플러그인 언어들은 문제를 해결하기 위해서 서로 협력해서 작업하기 때문에 플러그인 언어들 사이에 제어 흐름과 데이터 흐름이 발생하게 된다. Argos 언어에서 사용되는 각 플러그인 언어들은 서로 다른 문법과 프로그래밍 패러다임을 지원하는 독립된 단위이고, Argos에서는 이처럼 독립적인 플러그인 언어들이 상호 작용하기 위해서는 메시지를 주고받기 위한 방법이 필요하다.

컴퓨터 하드웨어에서 CPU, 메모리, 주변기기들은 밀접한 관계를 가지면서 서로 협력하여야 한다. 이때 각 요소들은 1 : 1로 연결하는 방법 대신에 버스라는 공통된 채널을 이용해서 서로 메시지를 주고받는다. 즉, 버스는 각 요소들을 1

:1로 연결할 때 발생할 수 있는 연결의 복잡성을 간단한 방법으로 해결할 수 있다. Argos 언어에서 플러그인 언어들 사이의 상호 작용은 버스 개념을 통해서 이루어진다. Argos 언어에서 사용되는 버스는 하드웨어 버스와 유사한 형태이며, 플러그인 언어들 사이의 상호 작용을 지원하기 때문에 언어 버스(language bus)라고 한다. (그림 4)는 Argos 언어에서 버스 구조이다.



(그림 4) Argos의 언어 버스

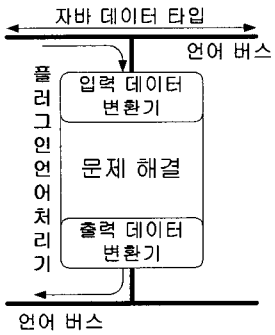
(그림 4)에서 회색으로 채워진 사각형은 Argos 클래스를 의미한다. 클래스에서 각 메소드들은 플러그인 언어들을 통해서 구현된다. 플러그인 언어들 사이의 상호작용은 메소드 호출을 통해서 이루어지고, 플러그인 언어들 사이의 데이터 흐름은 언어 버스를 통해서 이루어진다. 언어 버스를 통해서 전달될 수 있는 데이터는 자바 데이터 타입만 가능하다. 따라서 각 플러그인 언어들은 자신만의 고유한 데이터 타입을 사용할 수 있지만, 외부와 통신하기 위해서는 항상 자바 데이터 타입으로 변환하여야 한다.

언어 버스를 사용하는 경우에 플러그인 언어들 사이의 자료형 변환의 복잡성을 줄일 수 있다. 만약 N개의 플러그인 언어가 사용되는 경우에 언어 버스가 없는 경우에는 N×(N-1)개의 자료형 변환을 고려해야 되지만, 언어 버스를 사용하는 경우에 N개의 플러그인 언어 자료형을 자바 자료형으로 변환하는 것만 고려하면 된다. 언어 버스의 또 다른 장점은 플러그인 언어로 작성된 메소드의 매개 변수와 리턴 값이 항상 자바 자료형이기 때문에 사용자는 사용된 플러그인 언어에 무관하게 클래스를 사용할 수 있다는 점이다. 이러한 특성은 Argos 클래스가 자바 클래스와 구별되지 않고, 실세계에서 사용될 수 있도록 한다.

4.2 플러그인 언어 처리기 구조

플러그인 언어들끼리 상호 작용은 언어 버스를 통해서 이루어진다. 이때 언어 버스를 통해서 이루어지는 메시지 교환에 대해서 고려할 사항이 있다. 플러그인 언어는 자신만의 고유한 문법 구조와 데이터 타입들을 사용할 수 있기 때문에 이러한 사항들이 올바른 메시지 교환에 영향을 줄 수 있다. 플러그인 언어의 고유한 문법은 플러그인 언어 내부에서만 사용되기 때문에 외부에 영향을 미치지 않지만, 데이터 타입은 다른 플러그인 언어에 메시지 전달을 통해 전달될 수 있기 때문에 외부에 영향을 미친다. 따라서 플러그인 언어들끼리 올바른 메시지를 주고받기 위해서는 표준화된 자료형에 관련된 내용들이 필요하다.

Argos 언어에서는 언어 버스에서 전달되는 모든 자료형은 자바 자료형으로 제한한다. 따라서 플러그인 언어들 사이에 전달되는 모든 데이터들은 모두 자바 언어의 데이터 타입과 호환되어야 한다. 언어 버스에서 전달되는 자료형들이 자바 데이터 타입이기 때문에 플러그인 언어에는 자바 언어의 자료형이 전달된다. 플러그인 언어 처리기는 전달받은 데이터를 입력 데이터 변환기를 통해서 언어 고유의 자료형으로 변환해서 사용할 수 있다. 변환된 데이터는 플러그인 언어에서 문제 해결을 위해서 사용되고, 결과값은 다시 출력 데이터 변환기를 통해서 자바 언어의 데이터 타입으로 변환되어서 언어 버스로 나가게 된다. (그림 5)는 플러그인 언어 처리기로 데이터가 전달되는 형태와 언어 처리기에서 언어 버스로 결과 값이 전달되는 것을 보여준다.



(그림 5) 플러그인 언어 처리기 구조

언어 버스에서 플러그인 언어 처리기로 들어오는 데이터를 임포트 데이터(import data)라고 하자. 임포트 데이터는 플러그인 언어 처리기에 의해서 자바 언어에서 플러그인 언어의 데이터 타입으로 변환된다. 플러그인 언어에 임포트되는 데이터는 매개 변수, Argos 클래스의 멤버 필드, Argos 클래스의 메소드 호출 결과 값이 될 수 있다.

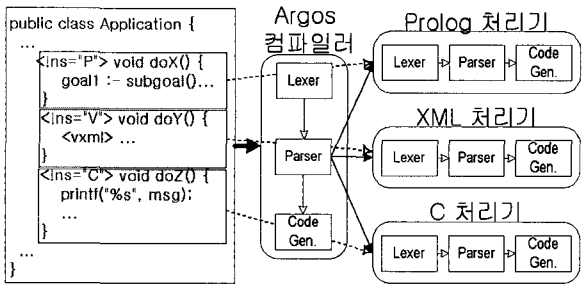
플러그인 언어 처리기에서 언어 버스를 통해서 외부로 전달되는 데이터는 익스포트 데이터(export data)라고 한다. 익스포트 데이터는 플러그인 언어 데이터 타입에서 자바 데이터 타입으로 변환되어서 전달된다. 익스포트 데이터의 예로는 Argos 클래스의 멤버 필드, 플러그인 언어의 리턴 문장, Argos 클래스의 메소드 호출에 사용되는 실 매개 변수가 있다.

클래스의 메소드는 멤버 필드와 매개 변수로 전달받은 값을 접근할 수 있다. 플러그인 언어에서는 멤버 필드를 접근하기 위해서 $ 문자를 멤버 필드 이름 앞에 붙여서 사용하고, 매개 변수를 접근하기 위해서는 $$ 문자를 접두어로 사용한다.

5. Argos 컴파일러 구현

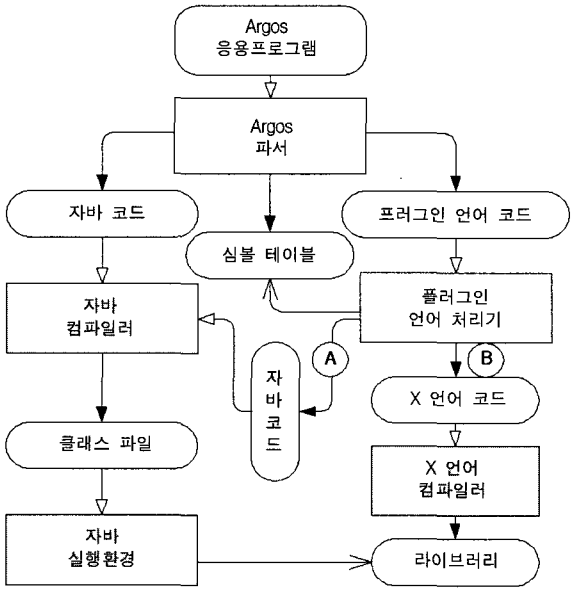
일반적인 프로그래밍 언어의 컴파일러는 어휘 분석, 구문 분석, 의미 분석을 수행하고, 최종적으로 목적 코드를 생성한다. 이때 어휘 분석과 구문 분석은 입력 프로그램이 주어진 언어의 규칙에 맞게 작성되었는지 여부를 체크하게 된다. 만약 언어에서 정의되지 않은 키워드를 사용하거나, 언어에서 정의되지 않은 문법을 사용하는 경우에 컴파일러는 에러를 발생시키면서 입력 프로그램을 처리할 수 없게 된다. 이러한 컴파일러 구조는 토큰과 문법이 고정된 언어에는 적합하지만, Argos와 같이 여러 개의 언어로 구성된 경우에는 사용할 수 없다. 따라서 일반 언어의 컴파일러와는 달리 Argos 컴파일러는 어휘 분석과 구문 분석 단계에서 임의의 언어를 처리할 수 있는 유연성이 있어야 한다.

하나의 소프트웨어 모듈로서 Argos 언어를 처리하는 컴파일러를 작성하는 것은 상당히 어렵기 때문에 본 논문에서는 DCO 모델에 따라 Argos 컴파일러를 작성한다. 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 처리기로 구성되어 있고, 새로운 플러그인 언어 처리기를 추가하는 경우에 Argos 응용프로그램에서 새로운 플러그인 언어도 사용할 수 있다. (그림 6)은 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.



(그림 6) DCO 모델을 따르는 플러그인 언어 파싱

(그림 7)은 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.

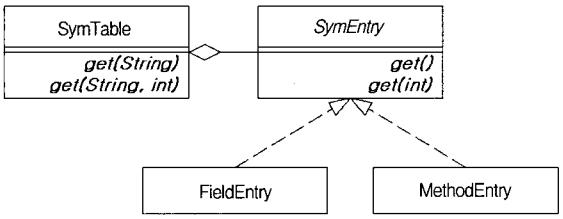


(그림 7) Argos 응용프로그램 컴파일 과정

플러그인 언어는 크게 2가지 형태로 처리된다. 첫 번째는 (그림 7)의 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 컴파일러를 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 파서는 입력 프로그램을 파싱하면서 플러그인 언어 처리기에서 필요로 하는 클래스의 멤버 필드와 메소드들을 심볼 테이블에 저장한다. 심볼 테이블은 플러그인 언어가 Argos 클래스의 멤버 필드와 메소드들을 사용하기 위해서 필요한 정보들을 관리한다. 따라서 플러그인 언어 처리기들은 심볼 테이블을 통해서 Argos 클래스의 멤버 필드와 메소드들에 대한 데이터 타입 정보들을 얻고, 플러그인 언어를 올바르게 처리할 수 있다. (그림 8)은 Argos 컴파일

러에 의해서 생성되는 심볼 테이블과 심볼 테이블 내용의 관계를 UML의 클래스도로 보여준다. 심볼 테이블에는 클래스의 멤버 필드와 메소드에 관련된 정보들이 저장된다. 심볼 테이블은 식별자를 이용해서 멤버 필드에 대한 정보를 얻을 수 있고, 메소드인 경우에는 식별자와 인덱스 값을 이용해서 리턴 타입과 매개 변수에 대한 정보를 얻을 수 있다. 자바에서는 멤버 필드와 메소드의 네임스페이스가 분리되어 있기 때문에 멤버 필드와 메소드 이름이 동일할 수 있다. 따라서 동일한 이름의 멤버 필드와 메소드를 구별하기 위해서 메소드 이름에는 "()" 접미어를 붙여서 사용한다.

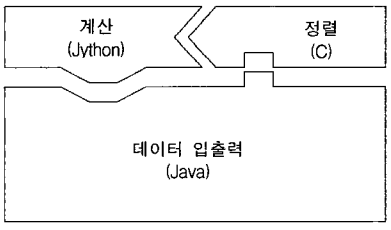


(그림 8) 심볼 테이블의 클래스 관계

Argos 컴파일러와 플러그인 언어 컴파일러들은 모두 LL(k) 파서 생성기인 JavaCC[30]를 이용해서 구현되었다. Argos의 문법은 J2SDK 1.4를 확장한 것을 사용하고, 자바 컴파일러는 SUN의 J2SDK 1.4를 사용하였다. Python 언어의 인터프리터는 자바로 구현된 Jython[13]을 사용했으며, Prolog 인터프리터는 JavaCC를 이용해서 구현하였다. 본 논문에서는 지면상 플러그인 언어 처리기의 구현에 관련된 내용은 기술하지 않는다.

6. Argos 응용프로그램 예

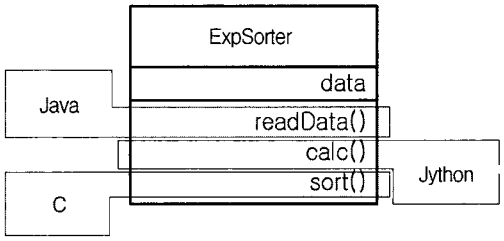
Argos가 사용되는 형태를 알아보기 위해서 간단한 Argos 응용프로그램을 작성하는 경우를 예로 들어보자. 응용프로그램은 콘솔로부터 수식을 입력받고, 계산된 결과를 정렬해서 출력해야 한다.



(그림 9) Argos에서 도메인 분석

소프트웨어 분석 설계를 수행할 때 문제의 도메인과 솔루션 도메인을 동시에 고려하면서 분석 설계하는 방법을 멀티패러다임 설계라고 한다[10]. 멀티패러다임 분석 설계를 수행하는 경우에 해당 문제는 (그림 9)와 같은 도메인으로 분류할 수 있을 것이다. 데이터 입출력은 자바 언어를 이용해서 수행하고, 동적인 계산은 Python 언어를 이용해서 수행한다. Python은 인터프리터 언어이고, 문자열 형태의 수식을 자동적으로 계산할 수 있는 기능을 제공하기 때문에 동적인 계산에 적합하다. 계산된 결과를 정렬하는 것은 C 언어를 이용해서 수행한다. 이것은 데이터를 정렬하는 C 언어의 라이브러리가 있다고 가정하기 때문이다.

도메인별로 분석된 내용은 (그림 10)과 같은 ExpSorter 클래스에서 여러 개의 플러그인 언어를 이용해서 설계 및 구현될 수 있다. ExpSorter 클래스에서 readData() 메소드는 자바 언어를 이용해서 구현되고, calc() 메소드는 Python 플러그인 언어를 이용해서 구현된다. sort() 메소드는 C 언어를 이용해서 작성된다.



(그림 10) ExpSorter 클래스에서 사용되는 플러그인 언어

ExpSorter 클래스의 구현된 내용을 살펴보면, 우선 계산을 수행하는 calc() 메소드는 <표 3>과 같이 한 줄의 Python 플러그인 언어로 작성할 수 있다. Python 플러그인 언어 처리기는 Python 코드를 자바로 구현된 Python 인터프리터인 Jython의 API에 맞는 코드로 변환해주는 역할을 한다.

〈표 3〉 Python 언어를 이용한 수식 계산

```
public <!ns = "jython"> double calc(String exp) {
    $return eval(exp) ;
}
```

실수 배열을 정렬하는 부분은 C 언어를 이용해서 작성할 수 있다. 만약 기존에 실수 배열을 정렬하기 위한 quicksort라는 함수가 라이브러리로 작성되어 있다면, 이 라이브러리를 C 플러그인 언어에서 직접 사용할 수 있다. C 플러그인 언어 처리기는 C 프로그램을 JNI 명세에 맞는 C 언어 코드로 변환시켜 준다. <표 4>는 C 플러그인 언어를 이용해서 수식이 계산되어서 저장된 실수 배열을 quicksort 라이브러리를 이용해서 정렬하는 것을 보여준다.

〈표 4〉 C 플러그인 언어를 이용한 정렬

```
public <!ns = "C"> void sort(double d[]) {
  int len = $$d.length ;
  quicksort($$d, len) ;
}
```

플러그인 언어들은 언어 버스 모델에 따라 서로 협력하면서 문제를 해결할 수 있다. <표 5>는 콘솔로부터 읽은 문자열을 calc() 메소드를 통해서 계산을 수행하고, 결과 값을 data라는 실수 배열에 값을 저장하는 방법을 보여준다. 데이터를 모두 읽고, 계산을 수행한 후에 정렬은 C 플러그인 언어를 통해서 작성된 sort() 메소드를 통해서 이루어진다.

〈표 5〉 플러그인 언어들의 상호 협력

```
...
  for (int i = 0 ; i < data.length ; i++) {
      while(true) {
          String exp = in.readLine().trim() ;
          if (exp.length() > 0) {
              data[i] = calc(exp) ;
              break ;
          ...
  sort(data) ;
...
```

응용프로그램 예에서 볼 수 있듯이 Argos 멀티패러다임 언어를 이용하는 경우에 문제를 객체지향 방법으로 분석 및 설계할 수 있고, 분할된 문제를 개발의 용이성과 라이브러리의 제공 여부에 따라 가장 적절한 패러다임과 언어를 선택해서 해결할 수 있다. <표 5>에서 볼 수 있듯이 플러그인 언어로 작성된 메소드도 일반 자바 메소드와 동일한 방법으로 호출해서 사용할 수 있기 때문에 기존 자바 개발자들이 쉽게 배우고 사용할 수 있다.

7. 결 론

컴퓨터 시스템이 발달하고, 플랫폼이 다양해지면서 소프트웨어의 요구 사항도 점차 복잡해지고 있다. 복잡한 소프트웨어를 개발하기 위해서 객체지향과 컴포넌트 개발 방법을 이용한 복잡도를 줄이고, 재사용성을 높이는 것도 중요하지만, 문제를 가장 적절한 프로그래밍 패러다임을 이용해서 해결하는 것도 중요하다. 이러한 필요성에 의해서 본 논문에서는 객체지향과 컴포넌트 방법론을 지원하면서, 멀티패러다임 프로그래밍도 지원할 수 있는 Argos 언어를 소개하였다. Argos 언어의 문법은 다른 멀티패러다임 언어와는 다르게 클래스의 메소드에 다른 언어의 문법을 추가할 수 있는 확장점을 제공함으로써 결합 방식(compositional approach)을 통해 클래스의 메소드를 다른 언어로 작성할 수 있는 방법을 제공한다. Argos는 자바 언어의 수퍼셋이고, Argos 클래스의 메소드는 자바 이외에 C, Python, Prolog 등의 다른 프로그래밍 언어를 이용해서 작성할 수 있다. 따라서 Argos는 문제 영역에 적합한 프로그래밍 언어를 선택해서 작성할 수 있기 때문에 주어진 문제를 좀 더 쉽게 해결할 수 있고, 기존에 작성된 라이브러리들을 그대로 재사용할 수 있다는 장점을 가지고 있다.

Argos에서 내장된 다른 프로그래밍 언어를 플러그인 언어라고 하고, 플러그인 언어들 사이에는 제어 흐름과 데이터 흐름이 발생한다. 제어 흐름은 메소드 호출을 통해서 이루어지고, 데이터 흐름은 언어 버스를 통해서 이루어진다. 언어 버스를 통해서는 자바 자료형만 이동할 수 있기 때문에 Argos 클래스의 메소드를 호출하는 경우에는 일반 자바 클래스와 동일한 방법으로 사용할 수 있다. 따라서 개발자는 Argos를 이용함으로써 기본적으로 객체지향 혹은 컴포넌트 프로그래밍을 사용하면서, 세부적인 문제들은 가장 적합한 프로그래밍 패러다임의 언어를 선택해서 해결할 수 있다.

Argos 응용프로그램은 여러 개의 언어들로 구성되어 있기 때문에 단일 컴파일러 모듈을 이용해서는 컴파일할 수 없기 때문에 DCO 모델에 따라 컴파일된다. 즉, 입력 프로그램은 Argos 컴파일러에 의해서 각 플러그인 언어로 작성된 코드에 따라 분할되고, 플러그인 언어를 처리할 수 있는 처리기에 전달되어서 컴파일된다. 플러그인 언어 처리기에서 생성된 코드는 Argos 컴파일러에 다시 전달됨으로써 전체적인 컴파일이 이루어진다.

현재 Argos는 함수형 패러다임을 지원하기 위해서 Python 플러그인 언어에서 Xoltar[9] 라이브러리를 사용한다. 이 라이브러리는 함수형 프로그래밍의 많은 기능들을 제공하지만, Lazy Evaluation과 같은 함수형 프로그래밍을 완벽하게 지원하지는 못한다. 따라서 Argos에서 함수형 패러다임을 완벽히 지원하기 위해서는 JVM 브리지[24]와 같은 방법을 이용해서 Haskell 혹은 SML과 같은 함수형 언어와 Argos를 연결하기 위한 연구가 진행되어야 한다.

Argos는 객체지향과 멀티패러다임 프로그래밍을 자연스럽게 결합함으로써 두 프로그래밍 패러다임의 장점을 이용해서 복잡한 소프트웨어를 효율적으로 개발하는데 도움을 줄 수 있을 것이다.

참 고 문 헌

[1] R. Nigel Horspool and Michael R. Levy, "Translator-Based Multiparadigm Programming," In *Journal of Systems & Software*, 23(1), pp.39-49, 1993.

[2] Bruce D. Shriver, "Software paradigms," IEEE Software, 1989.

[3] Timothy A. Budd, *Multiparadigm programming in LEDA*, Addison-Wesley, 1995.

[4] Kam Wing Ng, Linpeng Huang and Yongqiang Sun, "A multiparadigm language for developing agent-oriented applications," In *TOOLS*, pp.18-26, 1998.

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

[6] CiLab, The OpenDoc White Paper, 1994.
[7] Clemens Szyperski, Dominik Gruntz and Stephan Murer, *Component Software, 2nd ed.*, Addison-Wesley, 2002.
[8] 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/.
[9] Bryn Keller, Xoltar Toolkit, available at http://www.xoltar.org/.
[10] James O. Coplien, *Multi-Paradigm Design for C++*, Addison-Wesley, 1999.
[11] Kam-Wing Ng and Chi-Keung Luk, "A Survey of Languages Integrating Functional, Object-oriented and Logic Programming," In *Journal of Systems Architecture*, pp.5-36, 1995.
[12] Hafedh Mili, Ali Mili, Sherif Yacoub and Edward Addy, *Reuse Based Software Engineering Techniques, Organization and Measurement*, John Wiley & Sons, 2002.
[13] Jim Hugunin, "Python and Java : The Best of Both Worlds," In *Proc. of the 6th International Python Conference*, 1997, available at http://www.jython.org/.
[14] Diomidis Spinellis, "Small Tools for Automatic Text Generation," In *Proc. of USENIX*, 1998, available at http://www.usenix.org/.
[15] Timothy A. Budd, "The Return of Jensen's Device," In *Proc. of MPOOL*, pp.45-63, 2002.
[16] Brendan Machado and K. M. George, "A Model for Multi-paradigm Systems", In *Proc. of Computer Science*, pp.221-227, 1991.
[17] J. H. M. Lee and P. K. C. Pun, "Object Logic Integration : a Multiparadigm Design Methodology and a Programming Language," In *Computer Languages*, 23(1), pp.25-42, 1997.
[18] Jong-Myung Choi, Jae-Woo Yoo, "Multiparadigm within a Class," In *Proc. of ISFST*, pp.143-148, 1999.
[19] Valentino Vranic, "Towards Multi-Paradigm Software Development," In *Journal of Computing and Information Technology*, 10(2), pp.133-147, 2002.
[20] Pamela Zave, "A Compositional Approach to Multiparadigm Programming," In *IEEE Software*, pp.15-25, 1989.
[21] Brent Hailpen, "Multiparadigm Languages," In *IEEE Software*, 3(1), pp.6-9, 1986.
[22] Daniel G. Bobrow, "If Prolog is the answer, what is the question," In *Proc. of 5th Generation Computer Systems*, pp.138-145, 1984.
[23] Diomidis D. Spinellis, *Programming Paradigms as Object Classes : A Structuring Mechanism for Multiparadigm Programming*, Ph.D. Thesis, Dept. of Computing, Imperial College of Science, Technology and Medicine, University of London, Feb., 1994.
[24] Java VM Bridge for Functional Languages, available at http://sourceforge.net/projects/jvm-bridge/.
[25] Jennifer Hamilton, "Language Integration in th Common Language Runtime," In *SIGPLAN Notices*, 38(2), pp.19-28. 2003.
[26] Jennifer Hamilton, "Interlanguage Object Sharing with SOM," In *Proc. of USENIX Conference on Object-Oriented Technologies*, 1996.
[27] Box D., *Essential COM*, Addison-Wesley, 1998.
[28] Jan Bosch, "Delegating Compiler Objects : Modularity and Reusability in Language Engineering," In *Nordic Journal of Computing*, 4, pp.66-92, 1997.
[29] Jan Bosch, *Layered Object Model Investigating Paradigm Extensibility*, Ph.D. Thesis, Dept. of CS., Lund Univ., Sweden, 1995.
[30] JavaCC-The Java Parser Generator, available at https://javacc.dev.java.net/.
[31] 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.
[32] Shaun-inn Wu, "Integrating Logic and Object-Oriented Programming," In *OOPS Messenger*, 2(1), pp.28-37, 1991.
[33] Anton Eliens, *DLP - A Language for Distributed Logic Programming Design, Semantics and Implementation*, John Wiley & Sons, 1992.
[34] Michael Hanus, "Distributed Programming in a Multi-Paradigm Declarative Language," In *Proc. of PPDP, LNCS 1702*, Springer-Verlag, pp.188-205, 1999.
[35] H. Ait-Kaci, "An Overview of LIFE," In *Next Generation Information System Technology*, Springer-Verlag, pp.42-58, 1991.

최 종 명

e-mail : jmchio@comp.ssu.ac.kr
1992년 숭실대학교 전자계산학과 학사
1996년 숭실대학교 전자계산학과 석사
2003년 숭실대학교 컴퓨터학과 박사
관심분야 : 멀티패러다임 언어, XML, 도메인 언어, 유비쿼터스 컴퓨팅

유 재 우

e-mail : cwyoo@comp.ssu.ac.kr
1976년 숭실대학교 전자계산학과(학사)
1985년 한국과학기술원 전산학과(박사)
1986년~1987년 코넬 대학교 객원 교수
1996년~1997년 피츠버그대학교 객원교수
1999년~2000년 한국정보과학회 프로그래밍언어 연구회 위원장
1983년~현재 숭실대학교 컴퓨터학부 교수
관심분야 : 프로그래밍 언어, 컴파일러, 인간과 컴퓨터 상호작용