액티브 네트워크 프로그래밍 언어 동향

Trend of Programming Languages for Active Network

나중찬(J.C. Na) 능동프로토콜연구팀 선임연구원, 팀장
손승원(S.W. Sohn) 네트워크보안연구부 책임연구원, 부장
박치항(C.H. Park) 정보보호연구본부 책임연구원, 본부장

액티브 네트워크는 새로운 프로토콜 및 기술을 네트워크 상에서 채택하여 운용하는 것을 용이하게 할 뿐만 아니라 사용자가 네트워크 상에 필요한 기능을 수행함으로써 네트워크 관리나 새로운 서비스의 제공을 능동적이고 유연하게 수행할 수 있도록 하는 새로운 네트워크 패러다임이다. 액티브 네트워크가 가지는 가장 대표적인 특징은 네트워크 노드에 대해 프로그래밍이 가능하다는 것이며, 이들 노드간에 전송되는 패킷은 이동 코드의 특성을 갖는다. 따라서 본 고에서는 액티브 네트워크를 위한 인터페이스 계층을 제공하는 프로그래밍 언어 이슈와 관련 연구 동향을 살펴보았다.

I. 서 론

인터넷의 급격한 확산과 인터넷을 이용하는 응용의 수가 급격히 증가함에 따라 인터넷을 이용하는 응용 및 사용자의 네트워크에 대한 요구는 점차적으로 복잡해지고 그 수가 증가하고 있는 상태이다. 이를 수용하기 위한 네트워크 모델은 사용자의 네트워크 요구 기능을 수행하는 프로그램 코드를 전송 및 실행함으로써 통신망에 새로운 서비스를 신속하고 경제적으로 도입하고 망 자원들을 보다 적절하게 활용할 수 있어야 하며, 이러한 목표를 두고 연구되고 있는 분야가 액티브 네트워크 분야이다[1],[6].

액티브 네트워크가 가지는 가장 대표적인 특징은 네트워크 노드에 대해 프로그래밍이 가능하다는 것이며, 이들 노드간에 전송되는 패킷은 이동 코드(mobile code)의 특성을 갖는다[14].

따라서 본 연구에서는 제Ⅰ장 서론에 이어 제Ⅱ장에서는 액티브 네트워크 프로그래밍 언어에 대해 정의를 한 후 제Ⅲ장에서는 액티브 네트워크를 위한 언어가 가져야 할 기능적 특징을 살펴본다. 제Ⅳ장에서는 현재 사용되고 있는 몇 가지 프로그램 언어 비교를 통해 액티브 네트워크 프로그램 언어 개발 사례를 살펴보고 제Ⅴ장에서 결론을 맺었다.

II. 액티브 네트워크 프로그래밍 언어 정의

'이동 코드'란 이종의 프로세서들에 대해서도 코드의 변경 없이 이식이 가능하며 각각에 대해 동일한 의미(semantics)로 동작할 수 있는 프로그램들을 의미한다. 또한 분산환경에서 통신 개체들간에 전송되어지는 '코드를 내포한 객체(code-containing object)'를 의미하며, 이러한 경우 이동 에이전트(mobile agents)로 불리기도 한다. 이렇듯 목적에 따른 다양한 정의가 가능하지만 이동 코드의 기본적인 필요성은 다음과 같이 요약될 수 있다.

○ 효율성(efficiency)

만약 원격지와의 반복된 상호 처리가 필요한 경우, 원격지로 프로그램 코드를 전송하여 원격지와 코드간의 상호 처리를 유도하는 것이 오히려 더욱 효과적이다.

○ 간결성과 유연성(simplicity and flexibility)

서버가 각종 응용 프로그램의 코드를 저장하여 클라이언트의 요구에 따라(on-demand) 이들 프로그램을 다운로드 하는 것이 네트워크 유지를 보다 간편하게 한다.

○ 저장용량(storage)

필요에 따라 프로그램을 다운로드 함으로써 각 클라이언트가 유지해야 할 저장용량을 훨씬 줄여줄 수 있다.

따라서 이동 코드란 이종의 네트워크를 통해 이동가능하며 목적지에 도착하면 자동적으로 실행되는 소프트웨어로 정의하며, 액티브 네트워크 프로그래밍 언어는 이동코드를 기술하는 것이다.

III. 액티브 네트워크 프로그래밍 언어 이슈

이동 코드가 가져야 할 기능적인 요구사항은 이동 코드가 사용되어지는 목적에 따라 조금씩 차이가 있겠지만 주로 다음과 같은 공통된 조건들을 만족시킬 수 있어야 한다[2],[9],[14].

○ 이식성(portability)

코드가 경유해야 할 방대한 네트워크 환경을 고려할 때, 개별 네트워크 단위의 토폴로지(topology) 및 구조에 따른 다양한 프로토콜 버전들을 코드 내에서 모두 실현한다는 것은 사실 불가능하다. 따라서 다양한 네트워크 환경을 충족하면서도 실행이 가능한 언어적 특성으로서 이식성이 요구된다.

○ 안전성(safety)

그 발생지가 불분명한 코드의 안전성에 대해서는 제한된 추측만이 가능할 뿐 전적으로 신뢰할 수가 없다. 따라서 최근의 프로그램 언어들은 포인터 사용의 제한 및 엄격한 타입체크 등을 통해 프로그램의 안전성을 극대화하고 있다.

○ 보안성(security)

네트워크의 노드를 경유하며 코드가 실행되는 과정에서 보안성의 문제가 빈번하게 발생할 수 있다. 일반적으로 안전성과 보안성의 경계는 불분명한데 이는 프로그램의 안전성 측면에서의 취약점이 불법적인 목적을 위한 공격에 이용될 때는 바로 보안의 문제로 연관되어지기 때문이다. 일반적으로 보안성에 대해서는 다음과 같은 네 가지의 관점을 고려해 볼 수 있다.

- 기밀성(confidentiality): 일반적인 의미의 개인정보의 보호를 의미
- 무결성(integrity): 비권한자에 의한 개인정보의 변경 불허를 의미
- 가용성(availability): 서비스의 가용성을 보장
- 검증성(authenticity): 신원의 검증에 의해 사용자의 신용을 확인

○ 효율성(efficiency)

프로그램 언어로서의 가장 기본적인 사항으로 기밀성 및 보안성 등에 대한 구현 부담의 최소화도 여기에 포함된다.

프로그램의 안전성을 검사하기 위해서는 어떤 고급 언어로 작성된 프로그램을 이동 코드 상태로 전송하기 전에 컴파일을 통해 프로그램의 에러를 체크하는 것이 가장 일반적인 방법이다. 하지만 이러한 컴파일의 결과가 과연 전적으로 신뢰할 수 있는지에 대한 문제는 항시 내포하고 있다. 최근에는 보다 엄격한 안전성 보장을 위해 다음과 같은 세 가지 방법이 제안되어 사용되고 있다.

- 암호화된 서명을 사용하여 프로그램 작성자에 대한 신뢰여부를 파악하는 방법이 있다. 예를 들면 신뢰할 만한 소프트웨어 제작사의 서명이 첨가된 프로그램의 경우는 일단 안전성이 보장된다고 생각한다.
- 위와 같은 서명을 코드의 컴파일러에 대해 적용하는 방법이 있다. 즉 프로그램의 제작자는 프로그램을 전송하기 전에 소수의 신뢰되는 컴파일 사이트 중 한 곳으로부터 자신의 프로그램을 컴파일한 후 이 사이트에서 제공하는 서명을 획득하여 네트워크를 통해 목적지로 전송하는 방법이다.
- 작성된 프로그램의 원시 코드는 컴파일을 통해 어떤 중간언어(intermediate language)의 형태로 전환되어 이를 이동 코드로써 네트워크를 통해 전송된다. 이러한 경우 중간언어로의 컴파일러는 안전성 체크에 대해 충분한 효율성이 전제되어야 한다.

여기에서 한 가지 주의할 점은 위의 세 가지 방법들은 결코 서로에 대해서 배타적이지 않다는 것이다. 즉 첫번째와 두번째 방법의 경우, 이를 혼용함으로써 보다 효과적인 안전성의 보장을 기대할 수 있다.

IV. 액티브 네트워크 프로그래밍 언어 개발 사례

지금까지 액티브 네트워크를 위한 이동 코드 언어들이 가져야 할 기본적인 특성을 언급하였다. 이 장에서는 이러한 특성을 만족하는 실제언어로 Java [15], O'Caml[17], PLAN[8],[10],[11], Safety-Net[7],[13], NetScript[12],[16] 및 Sprocket [3]-[5]에 대한 언어적 특징과 안전성 및 보안성의 측면을 분석해 보았다.

1. Java

Java는 Sun Microsystems사에서 제작한 객체지향 언어이며, 이동성 및 보안에 주안점을 두었다. 특히 Java로 제작된 애플릿은 이동 코드의 좋은 예로써 네트워크를 통해 자동적으로 다운로드되며 해당 웹 페이지에서 실행되는 아주 작은 크기의 응용프로그램으로 간주될 수 있다.

가. 언어적인 측면

Java는 기본적으로 C++을 기반하였지만, C++가 가지는 특징 중 이동 코드로써의 불필요한 멎멎요소들을 제거하였다. 제거된 요소들에는 C/C++에서 사용되어온 포인터 연산, 비교적 무제한적인 캐스트(cast) 변환, 공용체, 비 구조적인 goto문, 연산자 재정의 및 다중 상속의 허용 등이 포함된다. C++에 대한 이러한 구조적 개선 외에도 자동적인 메모리 관리기능이 새로이 추가되었으며, 동적 메모리 할당을 보다 간단하게 구현할 수 있도록 하고 있다.

Java에서는 '*interface*'라는 독특한 타입 방식을 제공한다. '*interface*'는 추상화된 메쏘드이며, 이와 연관된 여러 변수들의 집합으로 정의된다. 이러한 '*interface*'를 구현한 클래스에 사용된 메쏘드들은 '*abstrac* '형으로 선언되며 필요에 따른 다양한 확장을 가능케 한다.

또한 '*package*'란 키워드를 제공하는데, 이를 통해 여러 클래스들과 인터페이스들을 하나로 그룹화할 수 있다. 이렇게 그룹화된 패키지는 일반적인 모듈 시스템과는 달리 개방형의 성질을 갖는다. 따라서 하나의 패키지는 그 패키지의 제작자가 설정하지 않은 정의를 이용하여 필요에 따라 얼마든지 새로운 기능들을 확장할 수 있게 된다.

Java에서는 GUI 및 네트워킹에 필요한 방대한 데이터 구조들을 라이브러리를 통해 제공하고 있다. 이러한 라이브러리를 이용해 작성된 각종 어플리케이션들은 자바 가상 머신이 동작하고 있는 여러 기종의 플랫폼상에서도 별도의 수정 없이 그대로 사용이 가능하다.

나. 보안적인 측면

Java는 현대적인 의미의 안전한 언어이다. 이는

언어에 적용된 설계 개념들과 다음과 같은 클래스들과 해당 어트리뷰트에 대한 접근규칙을 통해 안전성을 지원하기 때문이다.

1) 클래스에 대한 접근 키워드

- *final* : 해당 클래스에 대한 서브클래스의 생성을 금지한다.
- *abstract* : 해당 클래스의 인스턴스의 생성을 금지한다.
- *private* : 어떤 클래스의 선언 범위를 해당 클래스가 속한 패키지 내로 한정한다.

2) 어트리뷰트에 대한 접근 키워드

- *private* : 해당 어트리뷰트에 대한 접근 권한을 어트리뷰트가 속한 객체 내부의 어트리뷰트들로 제한한다.
- *default* : 해당 어트리뷰트에 대해서는 동일 패키지 내의 어떤 어트리뷰트들도 접근이 가능하다.
- *protected* : 어떤 어트리뷰트에 대한 접근을 그 어트리뷰트가 속하는 클래스의 서브클래스까지 허용한다.
- *public* : 해당 어트리뷰트에 대한 접근을 무제한으로 허용한다.

실제로 하드웨어 자원들에 대한 인터페이스는 이러한 접근 규칙을 통해 무분별한 접근으로부터 보호된다. 또한 파일 시스템 및 네트워크 접근과 같이 동적인 접근 제어를 필요로 하는 자원들에 대해서는 *SecurityManager*라 불리는 보안관제 모니터에 의해 제어된다. 이 외에도 보안과 관련된 메쏘드는 *final* 타입으로 선언함으로써 임의의 어플리케이션 혹은 애플릿에 의해 변경되는 것을 방지한다. 가상머신 차원에서는 Java 코드에 대한 정적인 체크뿐만 아니라 실행 시 에러체크를 제공함으로써 안전성을 지원한다.

다. 네트워크 측면

네트워크 상을 통한 클래스의 로딩은 '*ClassLoader*'라 불리는 클래스의 객체에 의해 실행된다. 이 객체는 프로그램의 시작 동안 생성되며, 이 후로는 자신뿐만 아니라 'java' 및 'sun'과 같은 몇몇 중요한 패키지들을 이동 애플릿에 의한 변경으로부터 보호한다. 이러한 보호기능 외에도 '*ClassLoader*'는 네트워크 상에서의 클래스들의 혼동을 막기 위해 유일한 이름 공간을 유지한다.

2. Objective Caml

Caml에서 비롯된 Objective Caml(O'Caml)은 선언적 기술언어로써 메타언어로 분류되며, 'MMM'이라는 웹 브라우저의 개발과정에서 이동 코드를 위한 언어로써 INRIA 연구소에서 개발되었다. MMM에서는 O'Caml으로 작성된 애플릿을 웹을 통해 동적으로 다운로드하여 실행할 수 있다. 또한 애플릿을 통해 사용자 메뉴에 새로운 항목을 추가하거나 새로운 컨텐츠를 위한 디코더를 첨가할 수도 있으며, 브라우저 내부 요소들(예, 내부캐시)에 접근할 수도 있다.

가. 언어적인 측면

O'Caml은 참조나 할당과 같은 특징들을 가지는 클래스 기반의 객체 지향 언어로, 모든 기능이 하나의 기능코어(functional core) 안에 통합되어 있다. Java와 마찬가지로 O'Caml에서도 클래스와 어트리뷰트들에 대한 접근 키워드를 제공하는데 대표적인 예로는 다음과 같은 것이 있다.

① *virtual* : 이 타입의 클래스에 대해서는 인스턴스를 생성할 수 없다.
② *closed* : 이 타입의 클래스에 대해서는 서브클래스를 생성할 수 없다.
③ *private* : 해당 클래스 내의 메쏘드들만이 접근할 수 있다.

나. 보안적인 측면

MMM에서의 애플릿은 안전한 표준 라이브러리

를 사용한다. 일반적으로 '안전한' 라이브러리는 확인되지 않은 사용자들로부터 유포된 모든 것들을 유입하지만, 받아들인 것들 중 선택된 것들만을 다시 유포한다. 즉 안전하다고 확신된 요소들은 사용자의 애플릿을 위해 그대로 유포하지만 위험의 여지가 있는 데이터 구조일 경우는 추상화된 형태로 유포한다.

MMM에서의 애플릿은 Java와는 달리 실행 전에 정적인 검사를 거치지 않는다. 오히려 도입된 모듈의 인터페이스에 대한 암호학적인 체크섬과 연관된 검증을 거치며, 이를 위해 애플릿 코드 컴파일을 위한 신뢰할 만한 사이트들을 제공하고 있다.

다. 네트워크 측면

O'Caml은 파일의 동적 링크를 위한 라이브러리를 포함하고 있다. 동적으로 로딩된 객체들은 자신이 공시한 함수들을 등록해야 한다. 또한 O'Caml은 애플릿에 대한 동적 링크 시 위험하다고 여겨지는 프리미티브들에 대한 사용을 제어할 수 있다.

3. PLAN

PLAN은 Upenn's PLANet 프로젝트에서의 패킷 프로그래밍 언어로써 개발되었으며, 특히 액티브 네트워크에서 가장 크게 대두되는 문제인 유연성과 안전성의 균형을 유지하는 데 중점을 둔 언어이다.

가. 언어적인 측면

PLAN은 액티브 네트워크에서 이동코드를 생성하기 위해 제안된 새로운 스타일의 언어이다. PLAN으로 작성된 프로그램은 아주 가볍고 또한 엄격하게 제한된 기능을 제공한다. PLAN 언어 자체가 가진 이러한 기능적 제한은 액티브 노드 내부에 장착된 강력한 기능을 가진 루틴의 호출을 통해 확장될 수 있다.

나. 보안적인 측면

네트워크를 공유한다는 것은 효율적이나, 안전성에 있어서 매우 치명적일 수 있다. 이러한 점을 고려하여 PLAN은 노드의 상태를 갱신하지 않는(stateless) 특징을 가지는 언어이다. 또한 PLAN은 포인터의 사용에 대해 안전하며 동시에 PLAN으로 작성된 프로그램들 간의 간섭에 대한 무결성을 보장한다.

일반적으로 액티브 네트워크에서는 네트워크의 안전성을 보장하기 위해 패킷의 소스에 대한 신원의 확인 및 검증의 필요성이 강력히 요구되는 반면에, 이를 처리하는 부담증가가 동반된다. 하지만 PLAN으로 작성된 프로그램의 경우는 그 기능성이 엄격하게 제한되어 있으므로 네트워크 상에서 문제를 발생시킬 가능성이 극도로 감소된다. 따라서 특별한 검증과정의 처리가 생략되는 장점도 지니게 된다.

다. 네트워크 측면

이동 코드는 원격지에서 수행되므로 프로그래머의 입장에서는 프로그램의 오동작의 원인을 결정하기가 쉽지 않다. PLAN에서는 프로그래머에게 프로그램의 동작에 대해 우선적인 보증을 제공한다. 즉 모든 PLAN 프로그램은 정적인 형태로써, 프로그램의 종료를 요청하는 서비스 루틴의 호출에 의한 강제적인 종료가 보장된다.

PLAN은 기본적인 예외사항에 대한 처리 메커니즘을 가지고는 있지만, 이것만으로는 시스템의 모든 오류를 처리할 수는 없다. 따라서 PLAN 프로그램이 어떤 오류를 발생시켰을 때, 프로그래머에게 이것을 알려주기 위해 PLAN에서는 두 가지의 특수한 오류 처리 메커니즘을 제공한다. 일단 '*abort*' 루틴을 이용하여 오류가 발생한 패킷을 소스노드로 다시 전송하여 그 프로그램을 확인하도록 한다. 그러나 어떤 오류의 경우 PLAN 프로그램의 내부에서 처리가 불가능할 경우도 있다. 이러한 경우에는 패킷의 헤더에 존재하는 '*handler*'라는 필드를 이용하여 오류를 처리한다. '*handler*' 필드에는 PLAN 프로그램의 내부에서 처리할 수 없는 오류나 예외가 발생했을 경우, 이를 처리하기 위해 프로그램의 소스 노드에서 제공하고 있는 함수의 이름이 포함되어 있다.

4. SafetyNet

가. 언어적인 측면

SafetyNet은 액티브 네트워크를 위하여 설계된 언어로 엄격한 타입체크 및 객체지향의 특징을 지니고 있다. SafetyNet 언어의 주된 목적은 액티브 네트워크에서 실행되는 분산된 어플리케이션들을 프로그래머가 사용할 수 있도록 하기 위한 것이다. 특히 강력한 타입체크 구조를 통해 SafetyNet으로 작성된 프로그램으로부터 네트워크의 무결성 파괴를 방지한다.

나. 보안적인 측면

액티브 네트워크용 언어들 중 SafetyNet과 가장 유사한 것으로는 앞서 언급한 PLAN을 들 수가 있다. 이들은 많은 면에서 유사한 특징들을 가지지만 가장 명확한 차이점은 이 두 언어의 네트워크 프로그래밍에 대한 접근방식에 있다. PLAN으로 작성된 제한된 기능의 프로그램들은 다른 일반적인 언어(예:O'Caml)로 구현되어 노드의 내부에 장착된 서비스 루틴들을 호출함으로써 보완된다. 이에 반해, SafetyNet에서는 PLAN에서와 같은 서비스 프로그램층의 의존성을 최소화하고, 대부분의 어플리케이션들을 SafetyNet 언어 자체로 구현하고 있다. 따라서 안전성 및 보안의 문제가 단지 SafetyNet 언어 자체의 오류검사에 의해 보장될 수 있다.

다. 네트워크 측면

일반적으로 분산 처리환경에서의 대부분의 객체들은 로컬 도메인 내에서 참조될 뿐, 원격지간의 참조는 이루어지지 않는다. 그렇지만 어떤 객체들은 마치 '공통된 자원(well-known resources)'처럼 동작하여 네트워크 상의 어디에서도 참조될 수 있는 특징을 갖는다. 이러한 객체에는 소켓(sockets), 서비스, 객체 레지스트리(object registries), 객체 데이터베이스(object databases) 등이 있다. SafetyNet에서는 이러한 객체들을 '*located objects*'라 부르고 있으며, 이들은 단일의 클래스 상태로 구현되어 있다. 이 클래스는 마치 프록시(proxy) 또는 래퍼(wrapper)처럼 동작하여 원격지에 저장되어 있는 또 다른 객체에 대한 접근을 제어해준다.

5. Netscript

NetScript 프로젝트는 네트워크 프로그래밍에 대한 접근 방식을 선언적 언어인 NetScript에 기반하고 있다. 즉, NetScript로 작성된 프로그램은 중간 노드에서도 마치 종단 노드처럼 동적이면서도 쉽게 수행할 수 있도록 하는 것을 목표로 하고 있다. 이를 통해 소프트웨어 벤더들은 혁신적인 어플리케이션 지원을 위해 중간 노드를 프로그래밍함으로써 보다 신속하면서도 유연한 서비스를 제공할 수 있게 된다. 예를 들어, 멀티캐스트 멀티미디어 어플리케이션 벤더들은 그 사용자들을 위해 종단 노드를 위한 어플리케이션 뿐 아니라 중간 노드에서의 라우팅 소프트웨어도 함께 제공함으로써, 사용자로 하여금 제공된 소프트웨어들을 종단 노드 뿐 아니라 중간 노드에서도 쉽게 사용하게 할 수 있다.

가. 언어적인 측면

NetScript는 통신기반의 작업을 위해 설계된 데이터 플로우 언어이다. 특히, NetScript 프로그램들은 패킷 스트림 상에서 동작한다. 이를 위해 NetScript 언어는 스트림 기반 계산 구조를 단순화하는 표준 프리미티브 집합을 제공한다. 이들 프리미티브들은 메시지 단위 동작(예를 들면, parse, flatten, split, join)과 메시지 스트림 단위 동작(예를 들면, multiplex, demultiplex)을 포함한다. 또한 NetScript는 단순한 객체지향의 원칙을 기반으로 한다. 따라서 프로그래머들은 기본적인 디폴트 연산자들을 그들만의 목적에 맞게 재구성할 수 있다. 일반적으로 NetScript는 다음과 같은 세 가지 관점에서 다른 언어들과 구별된다.

① NetScript는 프로그래밍 가능한 네트워킹 장치

의 보편적인 추상화를 제공한다. 포스트 스크립트는 보편적인 프린팅 장치에 대한 추상화를 제공하는 하나의 예이다.

② NetScript는 동적 언어이다. 이것은 프로그램이나 디바이스가 기존 프로토콜 수행을 방해함이 없이, 네트워크부터 제거되거나 추가될 수 있다는 것을 의미한다.

③ NetScript의 계산 모델은 데이터 플로우에 기초한다. 이 모델은 기존의 제어 플로우 모델과는 매우 다르다. 데이터 흐름 모델에서는, 시스템의 데이터 흐름이 각 프로세스의 계산을 조종하는 동시 발생적인 통신 프로세스들의 집합으로 구성된다.

나. 네트워크 측면

패킷에 포함된 최소한의 NetScript 캡슐 헤더들은 패킷이 속하고 있는 데이터 스트림에 대한 기본적인 구분자를 제공한다. 어떤 패킷이 가상 네트워크환경(Virtual Network Environment, 이하 VNE)에 도착하면, 헤더에 실린 정보에 의해 패킷이 처리될 수 있는 프로그램으로 전달된다. 이러한 통신 모델은 기본적으로 Milner의 'CCS semantic' 모델과 아주 유사하다. 이 통신 모델은 복수의 프로그램들이 하나의 데이터 스트림을 동시에 처리하는 것을 허용하고 있다. 이러한 프로그램들은 글로벌 라우팅(global routing)을 수행하거나 관리목적에서의 데이터 스트림의 통계자료 수집 또는 침입감지를 위한 패킷 모니터링 및 필터링 등에 사용될 수도 있다. 이때 NetScript 언어는 이러한 글로벌 라우팅 함수들은 기본적으로 내장되어 있지 않다. VNE에 의한 통신 서비스들은 전적으로 지역적(local)이다. 단지 이웃 VNE와의 상호작용을 허용함으로써 글로벌 라우팅을 구현할 뿐이다.

6. Sprocket

BBN's 스마트 패킷(Smart Packet) 프로젝트는 네트워크 관리의 성능과 유연성을 향상시키기 위해 액티브 네트워크 환경에서 사용할 수 있는 이동형 프로그래밍 언어인 Sprocket 언어와 Spanner 언어를 개발하였다. Sprocket 언어는 포인터 제거와 내장된 MIB 접근 타입 등을 갖는 C 언어와 같은 고급 언어이며, Spanner는 CISC 어셈블리 언어이다. Sprocket 코드는 Spanner 코드로 컴파일되며, 다시 기계에 대해 독립적인 바이너리 엔코딩 형태로 번역되어 액티브 패킷에 넣어지게 된다. Sprocket 언어의 설계에는 다음과 같은 사항들이 고려되었다.

- 정밀성(compactness)
- 보안성(security)
- 동적 링크, 외부 함수와의 인터페이스

가. 언어적인 측면

C 언어와 같은 고급 언어인 Sprocket 언어가 갖는 문법 특징은 다음과 같다.

① C의 문법과 키워드를 기초로 하였지만 스마트 패킷에서 불필요하다고 여겨지는 점들은 제외하였다. 대표적인 예로는 나열자(enumeration), 타입정의(type define) 및 구조체와 공용체를 들 수 있다.

② Sprocket으로 작성된 프로그램 어디에서든 C++ 스타일의 주석 및 선언형을 기술할 수 있다.

③ 참조나 해당 값에 의해 파라미터가 전달된다.

④ C 에서의 *int, short*, 혹은 *char*와 같은 타입들은 타입의 크기 및 부호를 직접 표기하는 방식으로 대체된다.

⑤ 부동소수점의 표현 값이 32비트 혹은 64비트까지 확장되었다.

⑥ *arrays, strings*, 그리고 *lists* 와 같은 타입들을 내부적으로 제공한다.

⑦ 이외에도 다음과 같은 새로운 타입형을 제공한다.

- packet 타입
- address 타입
- smart packet identifier 타입

• MIB context 타입

address 타입과 packet 타입은 Sprocket이 IPv 4, IPv6, 액티브 네트워크 등 어떠한 라우팅 환경에서도 동작될 수 있게 하는 추상적인 타입이며, 각각의 내용들은 자동적으로 환경에 맞는 특정한 형식을 취하게 된다. 대분부의 라우팅 환경은 동일한 개념(예를 들면 소스 주소, 목적 주소 등)을 이용하기 때문에, 이러한 추상적인 타입은 개념들을 처리하는 메쏘드들(예, “get source address”, “set source address” 등)을 포함한다.

나. 보안적인 측면

스마트 패킷에서는 다음과 같은 몇 가지 안전성의 요소를 고려하여 이를 Sprocket 언어의 설계에 반영하였다.

① 불안전한 연산의 배제
파일 시스템 접근 또는 시스템 호출 또는 권한을 갖지 않는 자의 데이터 접근을 수행 코드에 포함하지 않도록 한다.
② OS 또는 실행 환경에 대한 폐해 절감
수행코드가 지나치게 오랫동안 수행되거나 자원을 독점적으로 사용하는 프로그램은 노드의 안전성을 위해 그 실행을 중지시킨다.
③ 인증된 사용자에게만 스마트 패킷 프로그램의 작성을 허용한다.

다. 네트워크 측면

스마트 패킷에서는 실행 코드의 크기를 경량화시키기 위해 액티브 패킷의 프로그램 크기를 제한하였다. 즉 매우 작은 크기로 인코딩된 프로그램을 생성하기 위해 프로그램 크기는 실제적으로 1kbyte 이하로 제한하였으며, 방대한 크기의 기능은 프리미티브로써 제공하고 있다. 코드의 크기를 줄이기 위해서는 프로그램의 원시 코드를 바이트코드로 컴파일하는데 이때 다음과 같은 요소들은 컴파일된 코드에서 배제된다.

① 모든 라이브러리 함수의 포함
② 코멘트 및 부가적인 에러체크 코드
③ 파일 접근 및 GUI와 같은 불필요한 호출 혹은 심볼 테이블

마지막으로 위와 같이 컴파일된 코드는 최종적으로 내용의 손실이 없는 압축 알고리즘을 거쳐서 패킷에 실리게 되며 각각의 액티브 노드에서는 여러 가지 기능 함수들을 포함한 라이브러리와 이들 패킷의 동적인 링크를 통해 보다 효율적인 패킷 실행을 제공한다.

V. 결 론

지금까지 액티브 네트워크에 사용되어질 이동 코드용 언어들의 특징적 요구조건과 개발 사례에 대해 살펴보았다.

제Ⅲ장에서 언급한 액티브 네트워크 프로그래밍 언어 이슈 중 이식성과 효율성은 프로그램 언어 수준에서 많은 논의가 되어 왔지만, 프로그램 언어에서의 안전성 및 보안성의 구현을 위해서는 근본적으로 다양한 언어들 중에서도 보안성과 안전성을 보장하는 언어만을 사용해야 한다는 점에서 선택의 제약을 받아 왔다.

그렇지만 제Ⅳ장에서 살펴보았듯이 최근의 다양한 프로그램 언어들이 엄격한 타입 체크, 제한된 포인터 사용, 자동화된 메모리 관리와 같은 저 수준에서의 각종 안전성을 지원하면서 사용자의 선택의 폭이 확대되고 있는 추세이다.

프로그램의 안전성을 검사하기 위해서는 어떤 고급 언어로 작성된 프로그램을 이동 코드 상태로 전송하기 전에 컴파일을 통해 프로그램의 에러를 체크하는 것이 가장 일반적인 방법이다. 하지만 이러한 컴파일의 결과가 과연 전적으로 신뢰할 수 있는지에 대한 문제는 항상 내포되어 있다. 따라서 보다 엄격한 안전성의 보장을 위해 암호화된 서명(crypto-

graphic signature)을 사용하여 프로그램 작성자에 대한 신뢰여부를 파악하는 방법, 프로그램의 제작자는 프로그램을 전송하기 전에 소수의 신뢰되는 컴파일 사이트 중 한 곳으로부터 자신의 프로그램을 컴파일한 후 이 사이트에서 제공하는 서명을 획득하여 네트워크를 통해 목적지로 전송하는 방법, 작성된 프로그램의 원시 코드를 컴파일을 통해 어떤 중간언어(intermediate language)의 형태로 전환하여 이를 이동 코드로써 네트워크를 통해 전송하는 방법 등이 검토되는 경향이다.

그러나 이러한 방법들은 결코 서로에 대해서 배타적이지 않다는 것이다. 즉 첫번째와 두번째 방법의 경우, 이를 혼용함으로써 보다 효과적인 안전성의 보장이 기대된다.

참 고 문 헌

[1] 이수형 외 3인, 액티브 네트워크 기술 동향, ETRI 주간기술동향, 제 996호, 2001. 5. 6.

[2] 박정민, 채기준, Active Network의 보안 기술 발전 전망, *Sigcomm Review*, Vol. 1, No. 1, Dec. 2000.

[3] BBN, Justifications for Various Sprocket and Spanner Design Decisions, *BBN Technical Memorandum No. 1222*, Sep. 1999.

[4] BBN, Overview of Languages for the Smart Packets Project, *BBN Technical Memorandum No. 1219*, Sep. 1999.

[5] BBN, Sprocket Language Description for the Smart Packets Project, *BBN Technical Memorandum No. 1221*, Sep. 1999.

[6] D. Tennenhouse and D. Wetherall, Toward an Active Network Architecture, *Comp. Commun. Rev.*, Vol. 26, No. 2, Apr. 1996.

[7] Ian Wakeman, Alan Jeffrey and Rory Graves, Tim Owen, "Designing a Programming Language for Active Networks," *submitted to Hipparch special issue of Network and ISDN Systems*, June 1998.

[8] Jonathan T. Moore Michael Hicks Scott Nettles, Chunks in PLAN: Language Support for Programs as Packets, Special Issue on Protocol Architetures for the 21st Century, Vol. 16, No. 3, April 1998, pp. 437 - 444.

[9] Konstantinos Psounis, Active Networks: Applications, Security, Safety and Architectures, *IEEE Communications Surveys*, First Quarter 1999.

[10] Michael Hicks, Pnkaj Kakkar, Jonathan T. Moore, Carl A. Gunter, Scott Nettles, PLAN: A Packet Language for Active Networks, ICFP 98.

[11] Michael Hicks and Angelos D. Keromytis. *A Secure PLAN.* In International Workshop on Active Networks, 1999, Submitted; available at http://www.cis.upenn.edu/~switchware/papers/secureplan.ps.

[12] S. da Silva, D. Florissi, and Y. Yemini, "Composing Active Services in NetScript," DARPA Active Networks Workshop, Tucson, AZ, March 9-10, 1998.

[13] Tim Owen, "Programmer's Guide to the SafetyNet Language," Dec. 1999.

[14] Tommy Thorn, "Programming Language for Mobile Code," INRIA, Mars, *ACM Computing Survey*, 1997.

[15] D. Wetherall, J. Guttag, and D. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *Proc. IEEE OPENARCH '98*, San Francisco, CA, April 1998.

[16] Y. Yemini and S. daSilva, *Towards programmable networks*. In IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Oct. 1996.

[17] Xavier Leroy, The Objective Caml system release 3.02: Part II. The Objective Caml language, http://pauillac.inria.fr/ocaml