코드 이동성에 기반한 동적 기능 합성을 지원하는 자율적 객체

김인규[+] ○ 홍장의[+] 배두환[+]
[+]한국과학기술원 전산학과

Autonomous Object to Support Dynamic Composition of Functionality based on Code Mobility

Kim, In-Gyu[+] ○ Hong, Jang-Eui[+] Bae, Doo-Hwan[+]
[+]Dept. of Computer Science, KAIST

요 약

자율적 객체(autonomous object)는 분산 시스템에서의 기능(functionality)에 대한 융통성과 동적 확장성을 제공하기 위해 적합한 패러다임이다. 코드 이동성에 근거한 기능의 동적 합성이 자율적 객체에 의하여 지원되면 사용자의 다양한 서비스 요구 충족과 네트워크 부하 및 서비스 응답 시간 최적화 등과 같은 잇점을 얻을 수 있다. 본 논문에서는 이러한 잇점을 제공하는 코드 이동성에 기반한 동적 기능 합성에 대한 연구로서 기존의 연구들에서 제안하는 동적 기능 합성을 확장하고, 확장된 합성 메커니즘을 실현하기 위한 언어적 구조체(language constructs)를 설계하였다. 또한, 자율적 객체가 수행되는 환경으로서의 플랫폼을 제안하고 적용 사례를 통하여 코드 이동성에 근거한 동적 기능 합성의 예를 제시하였다.

1. 서론

인터넷 환경의 급성장에 따라 대규모 분산 시스템에서의 기능에 대한 융통성(flexibility) 및 동적 확장성(dynamic extension)에 대한 요구가 점점 커지고 있다. 기존의 분산 시스템을 위한 공학적 패러다임 - 예를 들면, 클라이언트/서버 패러다임 - 들은 위와 같은 요구사항들을 만족하기에는 부족하기 때문에 새로운 네트워크 컴퓨팅 패러다임들이 요구되었으며, 이러한 요구에 의해 코드 이동성(Code Mobility)에 기초한 패러다임들이 등장하게 되었다. 코드 이동성이라 함은 실행 가능한 코드가 특정 기능을 수행하기 위하여 논리적으로 또는 물리적으로 분산된 컴퓨팅 시스템을 옮겨다니는 것을 의미한다[1].

코드 이동성에 기초한 패러다임의 예로서 Code on Demand, Remote Evaluation, Autonomous Object(AO) 들을 들 수 있다[4]. 이들 중에서 AO 패러다임은 자율적 객체가 각 노드들을 이동하면서 자신이 필요한 기능을 동적 합성(dynamic composition)함으로써 분산 시스템에서 요구하는 기능의 융통성과 동적 확장성을 제공할 수 있다. 코드 이동성에 기반한 동적 기능 합성은 사용자 서비스의 요구사항을 충족시키기 위해 필요한 기능을 동적으로 합성할 수 있고, 새로운 버전의 기능을 쉽게 업그레이드 할 수 있으며, 원격지에 있는 기능을 재사용할 수 있을 뿐만 아니라 네트웍 단절과 같은 상황에도 잘 대처할 수 있다는 잇점을 제공한다

자율적 객체에서의 코드 이동성에 기반한 동적 기능 합성에 관한 연구들이 기존의 몇몇 측면에서 부분적으로 이루어져 왔다 그러나, 이러한 연구들은 분산 시스템에서 요구하는 융통성과 동적 확장성을 제공하기에는 부족한 것으로 보인다. 다시 말해서, 자율적 객체가 네트워크상에서 이동하면서 논리적 혹은 물리적 원격지의 기능을 이동시켜 합성하거나 합성된 기능을 동반하여 네트워크를 이동하는 것에 대한 연구는 거의 이루어지지 않고 있다.

따라서, 이 논문에서는 자율적 객체에서의 코드 이동성에 기반한 동적 기능 합성을 지원하기 위해 동적 기능 합성 메카니즘에 대한 확장된 개념을 제안하고, 이를 지원하기 위한 언어적 구조체(language constructs)를 설계하였다. 또한, 본 연구를 통해 제안하는 자율적 객체의 동적 기능 합성 메카니즘을 지원하는 자율적 객체 플랫폼을 제안하고 적용 사례를 통하여 이러한 메카니즘을 이용하는 예를 제시하였다.

2. 관련 연구

코드 이동성에 기반한 동적 기능 합성을 지원하는 자율적 객체에 대한 여러 연구들이 있다. 이러한 연구들 중에서 대표적인 두 연구, MESSENGERS[5]와 AGLETS[6]에 대하여 알아 보고 이들이 코드 이동성에 기반한 동적 기능 합성을 어느 정도 지원하고 있는지에 대하여 알아 보고자 한다.

2.1 MESSENGERS

MESSENGERS는 자율적 객체의 개념에 기반한 범용 목적의 분산 컴퓨팅을 위한 시스템이다[5] MESSENGERS를 제안한 Bic, et al 은 동적 기능 합성에 대하여 크게 4가지 수준으로 제안하고 있는데[2], (1) 자율적 객체가 동적인 기능 통합 능력 갖지 못하는 None 수준, (2) 현재 노드에서 새로운 프로세스를 발생시켜(spawning)

기능을 동적으로 통합하는 Process 수준, (3) 현재 노드에 존재하는 기능을 자율적 객체의 행동안으로 통합하는 Resident function 수준, (4) 다른 노드의 기능을 현재 노드로 가져와(carrying) 동적으로 통합하는 Carried function 수준이다. 이중에서 코드의 이동성을 근간으로 하는 Carried function 수준은 LAN 환경하에서 네트워크 파일 시스템에 의해 지원하고 있기 때문에, 자율적 객체내에서의 언어적 표현에 의해 코드 이동성을 지원한다고 볼 수 없다.

2.2 AGLETS

AGLETS[6]은 모발 자바 에이전트 즉, aglet을 지원하는 플래폼이다. Aglet은 자바 애플릿(applet)이 Web browser에 의해서 실행되듯이 aglet 서버에 의해서 실행된다 AGLETS에서는 코드 이동성에 기반한 동적 기능 합성을 지원하기 위한 환경을 제공하기는 하나, 명시적인 방법으로 제공되는 것은 아니다. 다시말해서, aglet은 자바로 작성되기 때문에 자바에서 지원하는 코드 이동성에 관한 여러 기능들을 이용할 수 있다[3]. 예를 들면, java.lang.ClassLoader는 원격지에서의 클래스를 가져오는(load) 것을 가능하게 한다.

3. 코드 이동성에 기반한 동적 기능 합성

자율적 객체에서의 코드 이동성에 기반한 동적 기능 합성이라 함은 그림 1 에서 보는 바와 같이 일반적으로 자율적 객체가 물리적 혹은 논리적인 위치에 있는 기능(method 수준의 기능 또는 객체 수준의 기능)을 자신의 행위안으로 동적으로 통합시킬 수 있는 능력을 말한다 그림 1 에서는 node B에 있는 자율적 객체 AO_1이 node A에 있는 기능 f1을 가져와서(carry) 사용할 수 있음을 보여주고 있다.

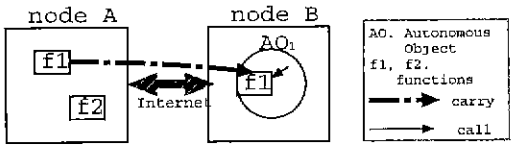

그림 1: 코드 이동성에 기반한 동적 기능 합성의 개념

위와 같은 자율적 객체에서의 동적 기능 합성 개념에 대해서 본 연구에서는 Bic, et al이 제안한 4가지 수준 중에서 Carried function 수준의 동적 기능 합성을 다음과 같이 확장하였다.

- Carried function based level : 명시적인 방법에 의해 자율적인 객체가 다른 노드에 있는 기능을 현재 노드로 가져와 동적으로 자신의 행동안으로 통합하는 수준
- Carried-All function based level : 자율적인 객체가 다른 노드에 있는 기능을 현재 노드로 가져와 동적으로 통합하고, 통합된 기능을 자율적 객체의 이동과 함께 동반하는 수준

Carried-All function based level은 자율적인 객체가 자신이 통합한 기능과 함께 동반하여 이동함으로써, 서로 다른 노드에서의 동일한 사용자 서비스에 대한 지원과 코드의 재사용 및 네트워크 단절에 대한 기능 수행의 지속성 등을 제공해 줄 수 있다 그림 2는 Carried-All function based level에 대한 개념을 나타낸 것이다.

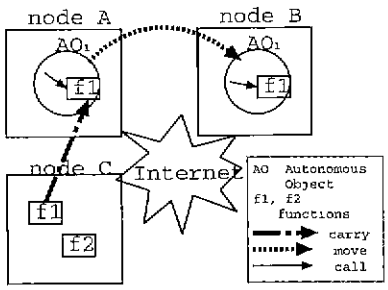

그림 2: Carried-All function based level을 지원하는 자율적 객체

4. 코드 이동성에 기반한 동적 기능 합성을 지원하는 언어적 구조체와 자율적 객체 플랫폼

자율적 객체에서 코드 이동성에 근거한 동적 기능 합성을 명시적인 방법으로 나타내기 위해서는 이를 지원하는 언어적 구조체가 필요하다. 이 언어적 구조체를 이용하여 명세된 자율적 객체는 코드 이동성에 근거한 동적 기능 합성을 지원하는 자율적 객체 플랫폼에서 수행된다.

4.1 언어적 구조체

먼저, Carried function based level의 구현을 위해 요구되는 구조체는 다음과 같다.

- *CFLoad(RemoteNodeLocation rnl, FunctionName fn)* - 원거리 노드 rnl에 있는 기능 fn을 자율적 객체의 행위안으로 동적으로 통합시킨다.
- *CFInvoke(FunctionName fn, Arg a)* - 자율적 객체가 기능 fn을 argument a를 주어서 실행시킨다. 이는 기존의 자율적 객체가 자신의 기능들을 실행시키기 위한 구조체(예를 들어, *Invoke()*)로 대체될 수 있다.

Carried-All function based level의 구현을 위해 요구되는 구조체는 다음과 같다.

- *CAFLoad(RemoteNodeLocation rnl, FunctionName fn)* - 원거리 노드 rnl에 있는 기능 fn을 자율적 객체의 행위 안으로 동적으로 통합시키고 자율적 객체가 이동할 때 같이 이동할 수 있도록 한다.
- *CAFDispose(FunctionName fn)* - 자율적 객체가 기능 fn을 자신의 행위 밖으로 버린다
- *CAFMove(RemoteNodeLocation rnl, CAFunctionName[] cafs)* - 자율적 객체가 이동할 때 *CAFLoad()*로 불러온 기능들(cafs)과 함께 이동한다.
- *CAFInvoke(FunctionName fn, Arg a)* - 자율적 객체가 기능 fn을 argument a를 주어서 실행시킨다. 이는 기존의 자율적 객체가 자신의 기능들을 실행시키기 위한 구조체(예를 들어, *Invoke()*)로 대체될 수 있다.

이상과 같은 언어적 구조체를 이용하여 명세된 자율적 객체의 간단한 코드 명세의 예는 다음과 같다.

```
(1)     Class AO_1 extend AO{
(2)        int a, b; String result; Function[] cafs;
(3)        ...
(4)        Do_Work(){
(5)           ...
(6)           CAFLoad("node C", f1);
(7)           CAFInvoke(f1, a);
(8)           ...
(9)           CAFMove("node B", cafs);
(10)          CAFInvoke(f1, b);
(11)          ...
(12)          PrintResult();
(13)          CAFDispose(f1);
(14)          ...
(15)       }
(16)       PrintResult(){ ... }
(17)       ...
(18)    }
```

위 코드는 그림 2에 있는 자율적 객체의 행동을 나타내고 있다. node A에 있는 자율적 객체가 node C에 있는 기능 f1을 가져와[(6)] 사용하고[(7)] node B로 이동한[(9)] 후 또 기능 f1을 사용하고[(10)] 다른 일들을 처리한[(11),(12)] 후 기능 f1을 버린다[(13)].

4.2 자율적 객체 플랫폼

코드 이동성에 기반한 동적 기능 합성을 지원하는 자율적 객체 플랫폼에 대한 프로토타입의 상위 레벨 아키텍처는 그림 3와 같다.

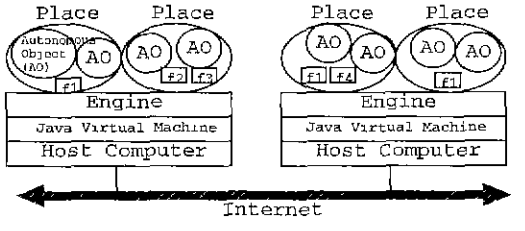

그림 3: 자율적 객체 플랫폼의 상위 레벨 아키텍처

Place는 물리적 노드(Host Computer)위에 있는 논리적 노드를 나타내고 기능(f1,f2,...)을 제공할 수 있다. Engine은 자율적 객체와 Place를 지원하는 역할을 한다

이 프로토타입은 인터넷 환경을 기반으로 하고 있으며, 자바를 이용하여 구현될 것이다. 따라서 동적 합성을 위한 기능은 객체 수준으로 제시될 것이다.

5. 적용 사례

코드 이동성에 기반한 동적 기능 합성을 이용하는 자율적 객체 어플리케이션으로 재사용 가능한 컴포넌트를 찾기 위해 여러 노드들에 존재하는 컴포넌트들을 테스팅하는 어플리케이션을 고려하였다. 다음은 사례에 대한 자율적 객체의 행위를 순차적으로 표현한 시나리오이다.

1. 자율적 객체가 컴포넌트 A에 대한 테스팅을 할 수 있는 기능 f1을 가지고[*CAFLoad(...,f1)*] 컴포넌트 A를 제공하는 여러 노드들을 돌아다니면서[*CAFMove(...)*] 컴포넌트 A에 대한 테스팅을 실시한다[*CAFInvoke(f1,...)*].
2. 자율적 객체가 여러 노드들을 방문하여 알아낸 재사용 가능한 후보 컴포넌트들을 보다 심도있게 테스팅하기 위하여 테스트 기능 f1을 버리고[*CAFDispose(f1)*] 테스트 기능 f2를 가지고[*CAFLoad(...,f2)*] 합격한 노드들을 찾아가 심도있는 테스팅을 수행한다[*CAFInvoke(f2,...)*]
3. 심도있는 테스팅 결과 최적의 재사용 컴포넌트를 갖고 있는 노드의 주소를 사용자에게 알려준다.

6. 결론 및 향후 연구 방향

본 논문에서는 자율적 객체에서의 코드 이동성에 기반한 동적 기능 합성에 대하여 연구하기 위하여 다음과 같은 세부 연구들을 수행하였다. 우선, 코드 이동성에 기반한 동적 기능 합성 레벨을 Bic. et al에 의해 제안된 개념으로부터 확장하였다. 그리고 이를 지원하기 위한 언어적 구조체를 설계하였으며, 자율적 객체의 실행 환경이라고 할 수 있는 플랫폼의 아키텍처를 제시하였다. 또한, 컴포넌트 테스팅의 사례를 통해 제시한 연구 결과의 적용 가능성을 설명하였다.

이러한 연구에 이은 향후 연구 방향으로, 기본적으로 플랫폼의 구현과 평가가 이루어져야 하며 또한, 보다 실제적인 환경에의 적용을 위해, 위임된(authorized) 노드만을 통한 기능 합성 방안과 원하는 기능의 위치를 찾아 주는 서비스 등이 연구되어야 할 것이다.

참고 문헌

[1] M. Baldi, S. Gai, and G Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," LNCS 1219. Springer-Verlag, pp. 13-26, April 1997.

[2] L. Bic, M. Fukuda, and M dillencourt, "Distributed Computing Using Autonomous Objects," *COMPUTER*, vol. 29, no. 8, pp. 55-61, August 1996.

[3] G Cugola, C. Ghezzi, G. Picoo, and G Vigna, "Analyzing Mobile Code Languages," LNCS 1222. Springer-Verlag, pp. 93-111, April 1997.

[4] A. Fuggetta, G Picco, and G. Vigna, "Understanding Code Mobility," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 24, no. 5, pp 342-361, May 1998.

[5] M. Fukuda, "MESSENGERS. A Distributed Computing System Based on Autonomous Objects," Ph.D DISSERTATION, Information and Computer Science, University of Carifornia, Irvine, 1997.

[6] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, 1998