이동 에이전트 시스템에서 AOP을 이용한 환경 특정 코드의 투명한 동적 적응기법

허성민[+O] , 안진호[++], 김차영[+], 황종선[+]
[+]고려대학교 정보통신대학 컴퓨터학과
{hursm[O], chayoung, hwang}@disys.korea.ac.kr

[++]경기대학교 정보과학부 전자계산학과
jhahn@kyonggi.ac.kr

Transparent and Dynamic Adaptation Scheme of Environment-Specific Codes in Mobile Agent Systems Using AOP

SungMin Hur[+O], JinHo Ahn[++], ChaYoung Kim[+], ChongSun Hwang[+]
[+]Dept. of Computer Science & Eng., College of Information and Communications, Korea University
[++]Dept. of Computer Science, College of Information Science, Kyonggi University

요 약

현대의 컴퓨팅 환경에서, 무선통신과 이동가능한 기기들의 등장 그리고 사용자의 요구조건의 변화로 인해서 소프트웨어는 동적으로 적응할 수 있어야 한다. 특히 새로운 컴퓨팅 패러다임으로 등장하는 이동 에이전트의 경우 어떠한 환경에서 수행될지를 모르기 때문에, 변화하는 상황에 대응한 적응력을 가지는 것이 필수적으로 요구된다. 이 논문은 다양한 변화에 대한 요구 사항들을 수용하기 위해서 이동 에이전트 시스템에서의 AOP을 이용한 환경 적응 코드의 투명한 동적 적응 기법을 제안한다. 본 논문은 동적인 상황에 대한 적응성을 역할의 분리를 통해서 이동 에이전트에 투명하게 적용하는 기법과, 다양한 정책을 적용함으로써 변화하는 환경에 대해서 성공적으로 작업을 수행하게 하는 기법을 보여줄 것이다.

1. 서론

현대 컴퓨팅 환경은 복잡해지고, 다양하게 변화하고 있다. 우선, 기존의 PC나 호스트 시스템에서부터 PDA, 이동전화 같은 플랫폼들이 새로이 나타나고 있다. 또한, 사용자와 기기의 이동성으로 인해서 사용할 수 있는 자원의 양과 환경이 계속적으로 변하게 된다. 소프트웨어를 개발하는 입장에서 이러한 변화하는 환경에 따라서 적응하는 것이 중요해지고 있지만, 동적으로 변화하는 환경의 정확한 정보를 미리 인지하고 적용하는 것은 불가능하기 때문에 이러한 소프트웨어를 개발하는 것은 어려움이 존재한다.

이러한 복잡성과 다양성들을 극복하는 하나의 대안으로 이동 에이전트의 개념이 등장했다[1]. 이동 에이전트의 특성상 다양한 환경으로 이동을 하기 때문에 수행하는 곳에서의 적응성은 필수가 된다. 언어적 측면에서 가상 기계의 개념을 도입한 JAVA의 경우와 이동 환경에서의 추상화 개념을 도입한 미들웨어들의 등장으로 인해서 어느 정도의 동질성은 보장해 주지만, 모든 자원과 환경에 대해서 동질성을 보장해 줄 수 없다. 또한, 동질성을 보장하기 위해서 추상화를 하게 되는 경우에는 수행 환경에서 사용할 수 있는 플랫폼의 기능이나 자원들을 희생해야 하는 경우가 발생하게 되었다. 이러한 단점들을 극복하기 위해서 다양한 기법이 논의되어 왔다.

소프트웨어 공학적 입장에서는 AOP(Aspect Oriented Programming)라는 기법으로 기능적인 부분들과 비기능적인 부분들(보안, 성능, 이동성 등)을 분리하는 것으로서 추가되는 변화사항들에 대한 다양성을 다루는 방법이 등장했다. 예를 들면, 먼저 응용프로그램 개발자가 수행에 필요한 부분인 기본 프로그램을 만들면, 나머지 부분들은 다른 분야의 개발자나 전문가가 자신의 분야에 해당하는 것을 첨가하는 형태로 작업하게 된다. 이렇게 작업을 하게 되는 경우 자신의 영역에만 충실하게 되면 되기 때문에, 소프트웨어의 성능 향상, 기능 추가 같은 유지 보수가 용이하게 되었다.

이동 에이전트의 분야에서도 이질성을 해결하기 위한 형태로서의 연구[2]가 존재했었지만, 이동 에이전트 프로그래머가 모든 변화부분을 미리 인지해야 하고, 단일한 정책밖에 적용할 수 없는 문제점들이 존재했다. 따라서, 이 논문은 AOP을 이용한 이동 에이전트 시스템에서의 환경 특정 코드의 투명한 동적 적응 기법을 제안한다. 먼저, 역할의 분리를 통해서 투명하게 이동 에이전트 프로그램에 다양한 환경에 대한 적응성을 가질 수 있도록 하였다. 또한, 정책 개발자가 다양한 정책을 적용함으로써 변화하는 환경에서 작업을 성공적으로 수행할 수 있도록 하였다.

2. 이동에이전트 시스템에서의 투명한 동적 적응 기법

2.1 개요

하나의 단일한 이동 에이전트를 기본적인 기능을 가지는 부분과 이질적인 환경에서 적응하는 두 개의 부분으로 나눌 수 있다. 전자를 핵심 부분이라고 하고, 후자를 적응 부분으로 정의할 수 있다. 다양한 환경에서 핵심 부분과 함께 환경을 인식하고 동적으로 적재하는 적응 부분에 해당하는 인터페이스를 가지고 다니는 이동 에이전트 시스템은 그림 1과 같이 묘사된다.

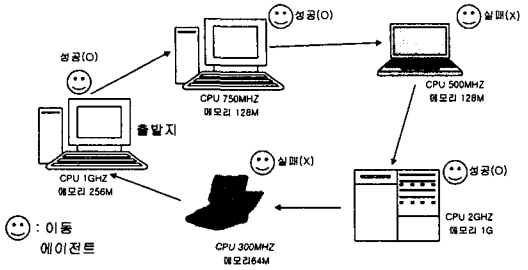

그림 1 단일정책이 적용된 이동 에이전트 시스템

실제 이동시에 변화하는 환경에 해당하는 적응 부분은 도착하는 시스템에서 환경을 인식한 후에 환경 특정적인 코드(메서드)를 동적으로 적재하게 되는 형태이다.

이 시스템에서의 문제점은 정확하게 모든 조건을 만족하는 정책(AND정책)만이 있다는 것이다. 5개의 노드에서 어떠한 알고리즘을 해당 환경에서 수행하는 예를 생각해보자. 각 노드마다 CPU 750MZ이상, 메모리 128M 이상인 경우에 수행되는 메서드와 CPU 1GHZ이상, 메모리 256M인 경우에 수행되는 메서드, 그리고 CPU 1.2GHZ, 메모리 512M이상인 경우에 수행되는 메서드 세 개가 설치되어 있다고 가정하자. 노드에 도착해서 CPU와 메모리의 상황을 인식한 후에 해당하는 알고리즘을 수행하기 위해서 필요한 환경 특정 메서드를 동적으로 적재하는 경우에, 그림 1의 경우 세 번째 노드와 다섯 번째 노드에서는, 정책상 정확하게 만족하는 구현이 존재하지 않기 때문에 작업을 수행하지 못하고, 에러로 처리하는 수밖에 없는 상황이 발생하게 된다. 만약 같은 환경에서 정확하게 일치하는 구현은 없다고 하더라도 최적의 근접한 정책을 제2의 정책으로 설정할 수 있다고 생각해보자. 그림 2의 경우, 세 번째와 다섯 번째 노드는 정책 1의 설정에 따르면, 원래는 수행이 불가능하지만, 정책 2의 경우는 성공을 보장하기 위해서 가장 근접한 상태 설정에 따르는 메서드(CPU 750MHZ, 메모리 128M 이상)를 수행하게 함으로써 성공을 보장한다. 정책을 유연하게 다중으로 선택할 수 있게 하는 그림 2의 시스템 형태가 가능하다면 이동 에이전트가 실패 없이 모든 수행환경에서 작업을 수행할 수 있을 것이다.

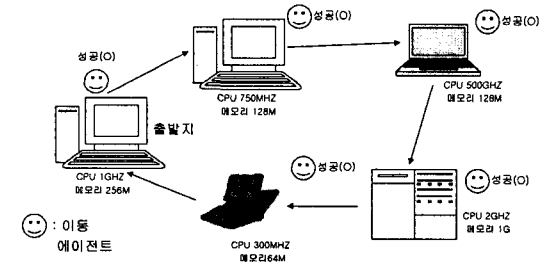

그림 2 다중정책이 적용된 이동에이전트 시스템

이 논문에서 제안하는 기법은 기존의 모델이 가지고 있는 대역폭의 절약과 핵심 부분과 적응 부분의 분리 같은 장점은 지닌 채로, 투명하게 다양한 정책을 적용하고 정적인 환경값들뿐만 아니라 동적인 환경값들을 다룰 수 있는 기법을 적용하였다. 이러한 기법들을 적용하기 위한 정의와 알고리즘은 이후의 절에서 자세하게 설명할 것이다.

2.2 투명한 정책의 적용

이동 에이전트 프로그램을 개발하는 과정에서, 이동 에이전트 프로그램 개발자는 변화하는 상황을 모두 인지해서 미리 준비할 수가 없다. 배포를 하는 시점에서 발견되는 사항이나 추가적인 요구조건들로 인해서 기존의 이동 에이전트 프로그램을 변경해야 하는 것이 요구된다. 미리 적응하기 위한 코드를 작성해두었다고 하더라도 설정값의 추가나 변경이 있는 경우는 기존의 코드 자체를 일일이 수정해야 한다. 따라서, 이동 에이전트 프로그램 개발자가 자신이 원하는 작업을 하면서도 변화하는 환경에 적응하기 위한 투명한 적응 기법이 필요하게 된다. 이동 에이전트 프로그램이 핵심 부분과 적응 부분으로 분리되어서, 적응이 필요한 부분이 인터페이스로 선언되어 있다면, 실제 구현은 여기에 맞추어서 코드(컴포넌트)를 개발하면 된다. 이러한 컴포넌트를 개발하는 역할은 해당 환경에 특정한 구현을 할 수 있는 컴포넌트 개발자가 하게 되고, 기존의 이동 에이전트 프로그램 중에서 변경사항을 반영하면서 정책이 설정된 연결 고리 역할을 하는 인터페이스를 제작하게 되는 것이 정책 개발자가 하게 되는 역할이다.

정책 개발자가 해야 하는 역할은 크게 두 가지이다. 첫 번째는, 변경이 요구되는 지점을 기존의 이동 에이전트 프로그램 코드에서 찾아내서 지정하는 것이다. 이러한 변경이 요구되는 지점은 플랫폼의 이질성, 이동성으로 인한 자원의 가용성 변화 같은 환경의 변화를 인지해서 적응해야 하는 부분들이 된다. 두 번째는, 변경 지정에 다양한 정책을 지정하는 것이다. 정책의 종류는 모두를 만족해야 하는 AND 정책, 하나만 만족해도 되는 OR 정책, 가중치를 적용한 우선순위 정책 등 다양하게 지정을 할 수가 있다. 그림 2에서 나오는 두 번째 정책은 가중치를 적용한 최소치의 적응 메서드를 찾는 정책이라고 볼 수 있다.

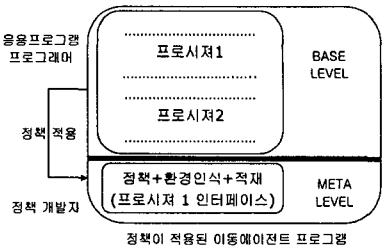

그림 3 정책 적용 과정

정책 개발자가 적용 지점과 정책을 설정하기 위해서 AOP 기법을 사용하면 기존의 이동 에이전트 프로그램이 그림 3의 형태로 변환되게 된다. 만약 이러한 변화를 단일한 프로그램으로 SWITCH - CASE 형태의 분기문을 사용해서 적용한다면, 크기가 커질 뿐만 아니라, 변경이나 추가 같은 유지보수가 어려워지게 되므로 투명하게 정책을 적용할 수 있는 AOP 기법을 사용한다. 이러한 적용 과정이 끝나게 되면, 기존의 프로그램에 변화에 적응하는 부분에 대한 인터페이스를 결합한 형태의 결과물의 형태로 최종적인 이동 에이전트 프로그램이 생성된다. 이 인터페이스에 들어가는 부분은 정책들, 인터페이스 명, 환경 인식 모듈 그리고 동적으로 적재를 하는 모듈이 들어가게 되는데, 자세한 기법은 다음 절에서 기술할 것이다.

2.3 동적 적응 기법

이동 에이전트 프로그램이 이동을 해서 새로운 수행 환경에 들어오게 되면 우선 현재의 환경을 인식해야 한다. 적응 프로시져마다 지정된 환경값들이 있는데, 해당 환경값들을 환경 모

니터를 이용해서 얻어오게 된다. 얻어진 환경값들과 지정된 정책들을 가지고 대응하는 적절한 메서드들을 코드 저장소에서 불러오게 되는 과정을 적응 전략이라고 할 수 있다. 동적인 적응 과정이 일어나게 되는 META LEVEL에서 인터페이스와 환경, 코드 저장소 간에 이루어지는 흐름은 그림 4와 같다.

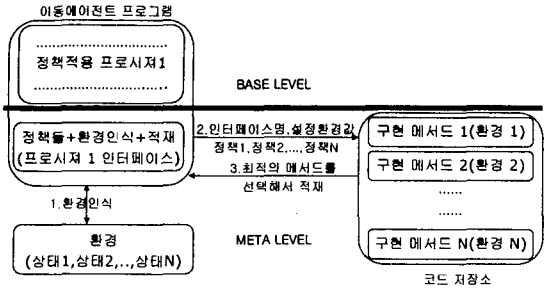

그림 4 동적 적응 기법

적응 전략을 적용하기 위해서 컴포넌트 프로그래머가 해주어야 하는 작업이 있다. 해당 인터페이스를 구현하는 것 뿐만이 아니라 그에 따르는 조건 함수를 구현해야 한다. 이 조건 함수는 지정된 환경상태의 값을 넣는 경우 참과 거짓을 반환하는 역할을 하게 된다. 예를 들면, 그림 2와 3의 경우 해당 인터페이스에 대해서 여러 개의 환경 특정 메서드를 작성을 할 때 CPU의 속도나 메모리의 용량을 인수로 받아서 그 메서드가 수행 가능한지를 판단하는 조건이 함께 첨부되어야 한다는 것이다. 메서드를 수행하는 알고리즘으로서 메모리가 256M가 필요한 경우에, 현재 환경 모니터가 128M의 값을 넘겨줄 경우 거짓을 반환하게 되는 식으로 수행되게 된다. 이러한 환경값에 대한 조건 함수들을 전부 확인한 후에, 결과치의 집합을 지정된 정책과 비교해서 적절한지를 판단하는 것이 적응 전략이 된다. 이에 대한 알고리즘은 그림 5에 나타난다.

```
L=0;                              //대응하는 메서드의 개수
INAME;                            //인터페이스 명
STATE[N];                         //관심 환경의 값들의 집합, N개
POLICY[M];                        //정책들의 집합, M개
METHOD[L];                        //대응하는 메서드의 집합, L개

L=FIND_METHOD(INAME);             //인터페이스에 해당하는 메서드 추출
IF (L>=1)                         //대응하는 메서드가 있는 경우에 수행
BEGIN
FOR I=1 TO M;                     //정책을 순서대로 반복한다.
   FOR J=1 TO L;                  //뽑힌 메서드를 순서대로 반복한다.
      FOR K=1 TO N;
            RESULT=COMPARE(STATE[K]); // 환경값들을 점검한다.
      NEXT K;
      IF (POLICY=MAXIMUM OR POLICY=MINIMUN); //정책을 점검한다.
            FINAL=COMPUTE(RESULT);      //결과치와 비교 저장
            IF (J=L)                    //마지막이면 해당 메서드 반환
                  RETURN METHOD[J];
            ELSE CONTINUE;              //아니면 다음 메서드로
      ELSE
            IF(RESULT=TRUE)
                  RETURN METHOD[J];//정책 적합시 메서드 반환

   NEXT J;
NEXT I;
END;
RETURN EXCEPTION;                 //적합한 메서드가 없으므로 예외 반환
```

그림 5 동적 적응 알고리즘

적응 전략을 통해서 적절한 메서드가 선택된 경우에는 이를 동적으로 적재를 해주어야 하는데, JAVA의 경우에는 코드를 동적으로 적재하는 기능이 포함되어 있기 때문에, 코드 저장소를 통해서 다양하게 적용하는 것이 가능하다. 이렇게 코드 저장소를 통해서 코드를 배치하는 것은 단일한 이동 에이전트 프로그램을 쓰는 것보다 크기를 절약해주기 때문에 확장성 면에서 유리하게 된다. 또한, 수행 속도의 향상을 위해서 정적인 환경값들에 대한 적응 메서드는 처음 한 번에 적재를 해주고 동적 적응 과정을 생략할 수 있다.

3. 관련연구

이동 에이전트 시스템에서 변화하는 환경에 대한 적응성을 지니도록 하도록 하는 연구는 기존에 여러 가지가 진행되어 왔다.

Sumatra[3]는, 자원 모니터의 개념을 도입해서 호스트 간의 네트워크 지연시간을 측정한 후 채팅 서버가 이를 활용해서 이동함으로써 반응시간을 줄이는 시나리오를 구현했다. 그러나, 이동 에이전트 프로그래머가 모든 상황에 대한 가정과 이에 대한 대응 행동을 섞는 SWITCH-CASE의 형태로 프로그램을 작성해야 하므로 추후에 환경이 추가되거나 변경을 하는 경우에 어려움이 생기는 문제점이 발생했다.

Brandt and Reiser[2]는, 이동 에이전트의 코드 저장소 개념 도입, 인터페이스의 자동 생성과 이질적 환경에서의 적응 등을 제안하고 구현했다. 하지만, 이동 에이전트 프로그래머가 변화 부분을 사전에 인지하고 있어야 하는 측면과 단일한 정책밖에 적용할 수 없어서 수행하는 환경에서 대응하는 클래스가 없는 경우 작업을 수행하지 못하고 실패하는 경우가 발생한다.

DAS[4]는 Meta Level의 개념을 도입해서 실제 작업 부분과 환경을 인식하고 정책을 설정해서 동적으로 적응하는 부분으로 나누는 시스템 모델을 제시하고, 이를 기술하는 언어인 LEAD++도 개발하였다. 범용적인 형태로 설계되었기 때문에, 이동 에이전트 시스템에 그대로 적용했을 때는 전체(BASE LEVEL + META LEVEL)가 단일하게 이동하는 형태로 되어서 유연성만큼 이동 에이전트 프로그램의 크기도 커지게 되어서 확장성에 문제가 생기게 된다.

4. 결론과 향후작업

이 논문은 AOP를 이용한 이동에이전트 시스템에서 환경 특정 코드의 투명한 동적 적응 기법을 제안한다. 역할의 분리를 통해서 이동 에이전트 프로그램에 투명한 적응성을 추가하고, 정책 개발자가 다양한 정책을 적용함으로써 변화하는 환경에서 작업을 성공적으로 수행할 수 있게 하였다.

현재 JAVA언어와 AOP기법을 사용한 프로토타입을 제작하는 중이고, 기존의 시스템과 동적 적응 전략 수립 시간, 적재 시간 등의 수행 시간의 측면과 코드의 크기 등을 비교할 예정이다.

참고문헌

[1] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," IEEE Transactions on Software Engineering, 24(5):352-361, May 1998.

[2] R. Brandt and H. Reiser, " Dynamic Adaptation of Mobile Agents in Heterogeneous Environments", G.P.Picco (Ed.): MA 2001. LNCS 2240. pp. 70-87, 2001.

[3] A. Acharya, M. Ranganathan and J. Saltz, "Sumatra: A Language for Resource-Aware Mobile Programs," Mobile Object Systems, J. Vitek and C. Tschudin, Eds., Springer Verlag, Apr. 1997, pp. 111-30.

[4] N. Aamano and T. Watanabe, " A Procedural model of dynamic adaptability and its description language", Proc.ICSE'98 International Workshop on Principles of Software Evolution, pp.103-106, 1998.