2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

Ad-hoc 망에서의 Jini Lookup Discovery 성능 개선

이진욱[0], 강대욱*
전남대학교 소프트웨어공학[0]
전남대학교 전산학*
{jinjin@moiza[0],dwKang@cs*}.chonnam.ac.kr,

Jini Lookup Discovery Improvement In Ad-hoc Network

Jin-Wook Lee[0], Dae-Wook Kang*
Dept. of Software Engineering[0], Dept. of Computer Science*
Chonnam Natl. Univ.

요 약

네트워크가 발전하면서 무선 환경에서 각종 장치들을 연결하고자 하는 요구가 늘어나고 있다. 무선 장치들은 전력소모를 줄여야하기 때문에 보다 많은 기능을 탑재하는 데에 어려움이 따른다. 특히, 실행중에 다른 장치를 인식하고 환경설정하며 실행상태를 관찰하는 것은 많은 에너지를 필요로 하는 복잡한 작업일 수 있다. Jini 기술은 장치들을 자동으로 인식하고 설정하는 데에 있어 강력하면서도 단순한 Service Discovery Protocol을 포함하고 있다. 그러나 Jini는 유선 네트워크를 기준으로 개발되어졌기 때문에 무선, 특히 Ad-hoc 네트워크 환경에 적용하기 위해서는 각 요소들이 어떠한 중계도 없이 독립적으로 서로를 인식할 수 있도록 개선되어야 한다.

Lookup Server가 작동을 멈추거나 룩업서비스를 받을 수 없는 지역에 있다면, 클라이언트는 서비스를 찾을 수 없고 서비스들도 새롭게 등록될 수 없다. 특히 이동성을 지닌 Ad-hoc 네트워크에서 룩업서비스의 존재는 보장받기 어렵다. 본 논문은 Jini를 기반으로 Ad-hoc 네트워크를 구축할 때 룩업서비스의 일부 기능을 클라이언트에게 분산시키고 다른 클라이언트와 정보를 공유할 수 있도록 개선된 Jini Discovery를 제안한다.

1. 서 론

지니는 썬 마이크로시스템즈에서 제안하고 있는 접속 기술로서 자바를 기반으로 유선, 모뎀, 무선 등의 다양한 방식으로 네트워크에 접속된 디바이스나 소프트웨어를 동적으로 상호 작용하게 하는 분산 미들웨어 기술이다. 네트워크상에 분산되어 있는 구성요소들을 탐지하여 동적으로 시스템을 구성하는 구조이며, 각종 디바이스를 통해서 네트워크에 접속이 이루어지면 시간과 장소에 상관없이 각종 서비스를 받을 수 있는 분산 네트워크 기술이다. 또한 하드웨어나 소프트웨어에 상관없이 지니를 채택하고 있는 디바이스들에 대한 인위적인 설치나 조작과 같은 일련의 절차를 배제할 수 있는 Plug & Play가 가능하다[1][2][3].

무선 Ad-hoc망은 중앙집중화된 관리나 표준화된 지원 서비스의 도움없이 임시 망을 구성하는 무선 이동 호스트들의 집합이다.

본 논문은 이러한 Ad-hoc망에서 기존의 Jini가 서비스를 찾는(discovery) 방식을 분석하고 무선 Ad-hoc 망에 맞게 개선안을 제시한다.

2. 연구배경

2.1 Ad-Hoc 망

무선망은 기존 유선망의 기반구조에 의존하느냐에 따라 Wirless network과 Ad-hoc network으로 나뉠 수 있다[9].

Ad-hoc 망은 이동 호스트로의 연결을 제공하기 위한 고정된 제어장치를 갖지 않으며, 각 이동 호스트가 라우터로 동작하여 이동 호스트로부터의 패킷을 다른 이동 호스트로 전달한다. Ad-hoc 망의 가

장 두드러지는 특징은 고정된 유선망에 대한 필요성을 최소화했다는 것이다. 다른 특징으로는 분산 계층 대응 모드(distributed peer-to-peer mode), 다중 홉 라우팅(multi-hop routing), 노드 배치의 상대적 잦은 변화 등을 들 수 있다.

Ad-hoc망은 그 특성상 임시 구성용 망이나 재해, 재난 지역이나 전장 등의 기반 시설이 갖추어져 있지 않은 환경에 적합한 것으로 인식되어 왔다. 최근에는 IETF의 MANet WG에서 이동, 무선 독립 IP부분에서의 인터넷 라우팅을 지원하기 위한 표준을 연구 중에 있다[5].

2.2 Jini discovery

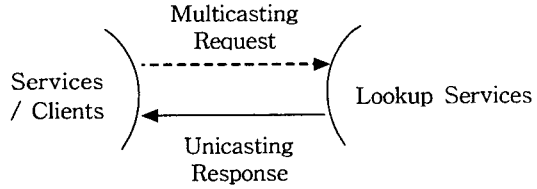
Jini의 기본적인 동작모델은 서비스제공자(Services), 서비스관리자(Lookup service), 서비스이용자(Client)사이의 상호작용으로 이루어진다.

서비스제공자(이하 서비스)는 프린터나 디지털카메라와 같이 특정 서비스를 제공하는 장치나 호스트이다. 서비스는 서비스관리자에게 서비스정보를 등록한다. 서비스관리자인 룩업서비스는 서비스에 관한 정보를 저장, 관리하고 클라이언트에게 제공한다. 서비스이용자인 클라이언트는 서비스를 이용하고자 하는 사용자나 응용프로그램을 말한다.

서비스와 클라이언트는 서비스를 등록하거나 특정 서비스를 요구하기 위해서 룩업서비스의 위치를 알아야 한다. 룩업서비스를 찾아 객체참조를 얻는 과정을 Discovery Protocol이라 한다.

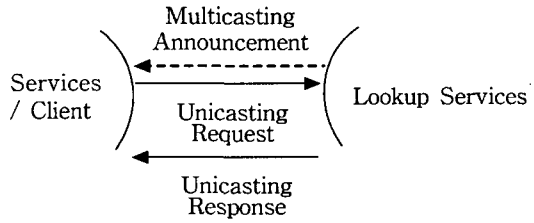
Discovery Protocol은 서비스, 룩업서비스, 클라이언트가 서로를 인식하는 과정으로 Jini 아키텍처의 핵심이 되는 부분이기도 하다. Discovery Protocol은 Multicast Request Protocol, Multicast Announcement Protocol, Unicast Protocol로 구분할 수 있다.

클라이언트와 서비스는 Multicast를 통해 룩업서비스의 주소를 알 수 있고, Unicast를 통해 직접 룩업서비스의 객체를 얻는다. 이 객체를 원격객체라고 하는데 원격으로 메소드를 수행할 수 있도록 한다.



<그림1: Multicast Request Protocol>

<그림1>과 같이 서비스는 자신을 룩업서비스에 게 등록하기 위해 룩업서비스를 찾는 메시지를 Multicast한다. 이것을 Multicast Request라고 하고 특정 서비스를 받고자 하는 클라이언트도 이를 이용해 룩업서비스를 찾는다. 이 메시지를 받은 룩업서비스는 메시지를 보낸 서비스나 클라이언트에게 응답한다. 이것이 Unicast Response 이다[1].

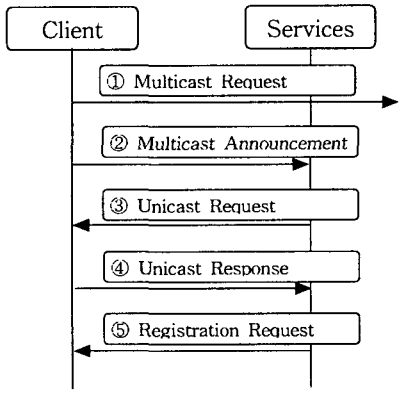


<그림2: Multicast Announcement Protocol>

또 다른 방법으로, <그림2>와 같이 룩업서비스는 모든 서비스나 클라이언트에 대해 자신을 알리는 Multicast Announcement 메시지를 주기적으로 보낸다. 이 메시지를 받은 클라이언트나 서비스는 그 룩업서비스에게 직접 원격객체를 요구(Unicast Request)하고 응답(Unicast Response) 받는다[1].

2.3 Peer Lookup

Jini Architecture는 기본적으로 룩업서비스를 필요로 한다. 논리적인 하나의 그룹에는 적어도 하나의 룩업서비스가 있어야 한다[1]. 그러나 서비스와 클라이언트가 이동성이 있는 Ad-hoc망에서 룩업서비스의 존재는 보장받을 수 없다[9]. Ad-hoc 망에서는 모든 호스트가 이동 가능하기 때문에 서비스, 클라이언트, 룩업서비스 모두 통신가능한 셀 영역안에 있음을 보장하기 어렵기 때문이다[4].



<그림3: Peer Lookup>

Jini에서는 룩업서비스가 없을 때 클라이언트가 서비스를 Discovery할 수 있도록 Peer Lookup이라

는 Mechanism을 제시하고 있다. 이것은 아래 <그림3>과 같다.

① 클라이언트는 룩업서비스를 찾는다. 이를 위해 Multicast Request 패킷을 보내고 응답을 기다린다.
② 아무런 응답도 받지 못한 클라이언트는 룩업서비스처럼 Multicast Announcement 패킷을 발행한다.
③ 서비스들은 룩업서비스에게 하듯이 Unicast Request를 발행하여 원격객체를 요구할 것이고
④ 클라이언트는 자신의 원격객체로 응답한다.
⑤ 서비스는 클라이언트 원격객체를 통해 등록을 요청한다. 클라이언트는 서비스들의 Proxy가 포함된 등록요청을 받게 되고 이것들 중에 원하는 서비스를 선택해 사용한다. 여기서 Proxy란 서비스 객체와 속성들을 포함한다.

그러나 클라이언트는 서비스 아이템들을 저장하지 않는다[1]. 만약 다른 서비스가 필요하면 Peer Lookup을 반복한다. 결국, Peer Lookup이 자주 일어날 수 있다.

뿐만 아니라 Peer Lookup으로 룩업서비스를 발견하지 못한 클라이언트는 다른 클라이언트들과 상호작용하지 않는다. 즉 클라이언트는 많은 노력에 의해 받은 가까운 미래에 필요할 수도 있는 서비스 항목들을 원하는 서비스를 선택한 후 폐기시킨다[1]. 이것은 추가적인 Multicast 메시지를 유발하고, 네트워크 트래픽을 발생시킬 수 있다. 만약, 가지고 있는 정보를 다른 클라이언트와 공유할 수 있다면 Multicasting되는 메시지를 줄일 뿐만 아니라 Ad-hoc망에서 유용할 것이다.

3. 제안기법과 구현

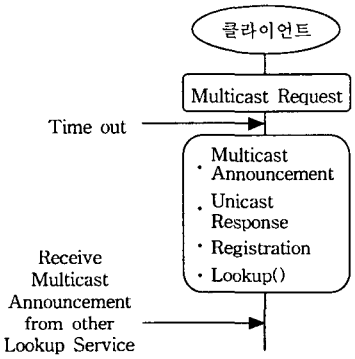
3.1 제안기법

Ad-hoc 망에서는 서비스들에 대한 정보를 중앙집중적으로 관리하고 클라이언트에게 이를 제공해주는 룩업서비스의 존재를 보장할 수 없다. 왜냐면, 룩업서비스를 중심으로 논리적인 Jini 연합체를 형성하는데, 각 요소가 이동성을 지닌 mobile node라면 서로가 통신가능한 영역 안에 머물러 있지 않을 가능성이 많기 때문이다.

제안하는 기법은 Peer Lookup을 확장하여 다음과 같은 개선을 추구한다.

첫째, 클라이언트가 룩업서비스의 도움없이 서비스를 얻을 수 있다. 둘째, 클라이언트가 임시 룩업서비스의 역할을 할 수 있다. 셋째, 다른 룩업서비스를 만나면 그 연합체에 소속된다.

룩업서비스를 찾지 못한 클라이언트는 스스로 룩업서비스의 기능을 수행한다. 다른 클라이언트로부터 룩업서비스를 찾는 Multicast Request를 받으면 이에 응답하여 자신의 원격객체를 보냄으로서 서비스 정보를 공유한다. 그러나 완전한 룩업서비스가 되는 것은 아니다. 계속해서 움직이고 있는 이 클라이언트가 룩업서비스의 Multicast Announcement를 만나면 더이상의 룩업서비스기능을 중단한다. <그림4>와 같다.



<그림4: 개선된 클라이언트>

즉, 확장된 Peer Lookup에서 클라이언트는 Multicast Announcement을 통해 서비스를 찾고 rease time동안 서비스 정보를 저장해둔다. 그리고 다른 mobile 클라이언트들의 Discovery 요구에 자신의 원격객체를 전달함으로써 정보를 공유한다. 여기서 rease time이란 Jini에서 서비스제공자와 사용자 간에 합의되는 서비스사용시간을 말한다.

기존의 Peer Lookup에서는 클라이언트는 원하는 서비스를 선택한 후, 많은 비용을 들여 얻은 서비스 항목들을 모두 폐기시킨다. 이 서비스 항목은 서비스 Proxy와 속성을 포함한다.

제안되는 기법에서 서비스의 등록요청을 받은 클라이언트는 구체적으로 다음과 같은 기능을 갖춘다.

① 서비스ID를 부여하여 서비스항목을 등록, 저장한다.
② lookup() 메소드로 원하는 서비스를 찾을 수 있다.
③ 저장된 서비스 항목들 중에 원하는 서비스가 없으면 다시 룩업서비스를 찾는다.
④ 다른 클라이언트로부터 룩업서비스를 찾는 Multicast Request를 받으면 응답하고 원하는 서비스 정보를 제공한다.
⑤ 서비스 항목들은 rease time이 만료되기 전에 갱신되지 않으면 폐기된다.

클라이언트는 룩업서비스가 수행하는 일부기능을 수

행하지만 다음과 같은 점에서 룩업서비스와 다르다.
첫째, 룩업서비스와 같은 Multicast Announcement를 정기적으로 수행하지는 않는다.
둘째, 클라이언트는 원하는 서비스를 이미 저장된 목록에서 일차적으로 검색한 후, 찾지 못했을 때에만 peer lookup을 재수행한다.
셋째, 다른 클라이언트나 서비스의 Multicast Request에 대해서는 룩업서비스로서 응답하고, 다른 룩업서비스의 Multicast Announcement를 받으면 클라이언트로서 행동한다. 이것은 룩업서비스가 많기 때문에 중복되는 서비스 검색을 없애기 위해서이다.

3.2 구현 및 테스팅

제안된 기법을 구현하기 위해 룩업서비스나 서비스를 수정할 필요는 없이 클라이언트만 수정한다.

구현은 간단한 "HelloWorld"문자열을 표준 출력해주는 서비스에 대해 룩업서비스의 중계없이 서비스를 얻을 수 있는 클라이언트를 개발했다. 테스팅을 위해 HelloWorld 서비스와 개선된 클라이언트, 비교를 위해 기존의 클라이언트, reggie 룩업서비스를 각기 다른 machine에서 순차적으로 수행시켰다.

HelloWorld서비스는 HelloWorld문자열을 표준출력하는 메소드를 구현했고, 기존의 클라이언트는 discovery한 룩업서비스 목록을 표준출력하는 코드를 구현했다. 테스팅의 마지막에 이용된 reggie 룩업서비스는 표준 Jini 룩업서비스로서 SUN사에서 개발한 것이다.

machine에 JVM을 설치하고 개선된 클라이언트와 서비스를 각각 다른 machine에서 실행시켰다. 이때, 클라이언트를 수행하기 전에 원격호출을 위해 rmi daemon을, 요구된 code를 download하기 위해 httpd를 수행시켰다. 또 서비스를 수행하기 전에 요구된 code를 전달하기 위해 httpd를 실행시켰다.

먼저, HelloWorld 서비스를 수행시켰다. 이후, 개선된 클라이언트를 실행했을 때, 이 클라이언트는 수행후 일정시간동안 룩업서비스의 응답을 기다렸고 아무런 응답없이 시간이 만료하면, 룩업서비스처럼 행동하여 HelloWorld 서비스를 얻는 데에 성공했다.

개선된 클라이언트의 수행을 확인 한 후, 개선되지 않은 클라이언트를 추가로 실행시켰다. 이때, 개선되지 않은 기존의 클라이언트에 대해 개선된 클라이언트가 룩업서비스로서 discovery 됐다.

마지막으로 reggie룩업서비스를 개시했을 때, 개선된 클라이언트는 더 이상 룩업서비스처럼 행동하지 않고 서비스를 얻기 위해 reggie를 찾는다.

4. 결론 및 향후 연구방향

이 논문은 Ad-hoc망이라는 특수한 환경에서 Jini discovery 의 개선방향을 제시하였다.

기존 Jini에서 서비스나 룩업서비스의 변형없이 Ad-hoc망 환경에 잘 적응하도록 클라이언트를 개선하였다. 구현된 클라이언트는 룩업서비스가 없이도 서비스를 찾을 수 있다. 어느 곳에 있든지 서비스를 얻을 수 있다는 점에서 이동성이 있는 Ad-hoc망에서 유용하고, 룩업서비스가 없는 곳에서 룩업서비스를 대신하기에 충분했다.

결과적으로, 룩업서비스나 서비스의 변형없이 Jini 클라이언트가 임시적으로 룩업서비스의 일부기능을 수행함으로써 Ad-hoc망의 특성에 잘 적응할 수 있었다. 그러나 한편으로는 클라이언트가 무거워진다는 점은 피할 수 없다. 즉, 룩업서비스의 기능을 갖추기 위해 클라이언트는 보다 많은 클래스를 포함하고 있어야 한다.

향후, Jini 클라이언트가 동시에 얼마나 많은 서비스를 발견할 수 있고, 얼마나 많은 다른 클라이언트에게 서비스정보를 제공해줄 수 있는지를 검증하여 임시 룩업서비스로서의 한계를 규명하고 개선할 필요가 있다.

5. 참고문헌

[1]. Sun Microsystems Inc., Jini Architecture Specification, available at http://www.sun.com/jini/specs.
[2]. W. K. Edward, Core Jini, 1999, Prentice Hall.
[3]. Sing Li, Professional Jini, 2000, WROX.
[4]. 김동완, 이성식,"이동AdHoc네트워크기술개요," 2000, available at http://ktwww.kotel.co.kr/itj.
[5]. The Internet Engineering Task Force, Mobile Ad-hoc Networks (manet), available at http://www.ietf.org/html.charters.
[6]. Sun Microsystems Inc., Jini Technology1.1 API Documentation; available at http://java.sun.com/products/jini/1.1/docs/api.
[7]. G. G. Richard III., "Service Advertisement and Discovery," 2000, IEEE internet computing.
[8]. Sun Microsystems Inc., A Collection of Jini Technology Helper Utilities and Services Specifications, 2000, available at http://www.sun.com/jini/specs.
[9] Michel Barbeau, "Service Discovery Protocols for Ad Hoc Networking," 2000, CASCON.