2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

JINT: Jini 인터셉터 설계 및 구현 방안

김중한, R.S.Ramakrishna
광주과학기술원 정보통신공학과
e-mail : {jini, rsr}@kjist.ac.kr

JINT: A New Service Selection Mechanism in Jini

Joong-Han Kim, R.S.Ramakrishna
Dept. of Information and Communication, K-JIST

요 약

Jini 는 분산 컴퓨팅을 위한 새로운 아키텍쳐이다. 리싱등을 포함한 다양한 분산 컴퓨팅을 위한 패러다임들이 Jini 에서 제시되었다. Jini 서비스를 이용하기 위해 대부분의 경우 Sun 사가 제공하는 Jini 라이브러리를 사용하여 서비스를 검색한다. 이러한 서비스 검색을 위한 라이브러리의 사용은 불투명한 서비스 선택정책 및 시스템 성능저하의 문제점을 가지고 있다.
본 논문은 이러한 문제점들을 해결하기 위해 Jini 를 위한 인터셉터인 JINT 의 설계 및 구현방안을 제시한다. JINT 의 사용은 투명한 서비스 선택을 가능케 해주며 네트워크 혼잡 및 과대한 시스템 리소스를 사용하는 서비스 프록시 다운로드의 문제점을 해결할 수 있다. 또한, JINT 의 이용에 대한 효율성을 알아보기 위해 기존에 제안된 Jini 부하분산을 JINT 를 사용하여 구현하는 방안을 제시한다.

서론

• Jini 아키텍쳐

Jini 아키텍쳐는 네트워크상에서 서비스의 정의,등록 및 검색을 지원하기위한 기반구조를 제시한다. Jini 의 주목적은 플러그-앤-워크(Plug-and-Work), 하드웨어/소프트웨어간 동등화, 즉각적 통신(Spontaneous Networking)과 서비스기반 아키텍쳐의 완성이다. Jini 는 이동성, 리싱, 분산이벤트등의 주 특성적 기술을 내포한다[1].

• Jini 프로그래밍 모델

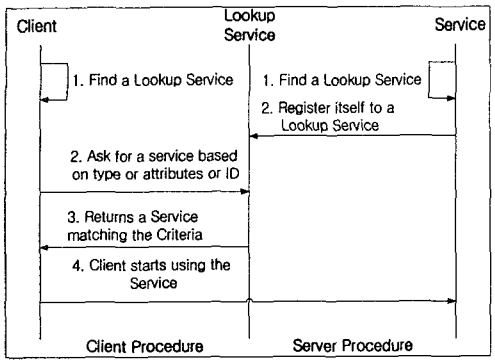
Jini 프로그래밍 모델은 다른 분산 컴퓨팅 모델과 달리 네트워크 연방(Federation)에서 서비스들이 등록, 검색되는 방식에 의존한다[2]. 서비스 등록, 이용을 위해 Jini 는 디스커버리 프로토콜(Discovery Protocol)을 이용한다. 디스커버리 프로토콜은 멀티캐스트 디스커버리(Multicast Discovery), 유니캐스트 디스커버리(Unicast Discovery), 멀티캐스트 어나운스먼트(Multicast Announcement)의 세가지가 사용된다.

• Jini 룩업 서비스

Jini 룩업 서비스(Lookup Service)는 djinn 상에서의 서비스 아이템 및 서비스 표현을 위한 중앙 저장소이다. 여기서 djinn 이란 Jini 기술 기반구조상에 참가하는 디바이스, 리소스, 및 사용자 등의 그룹을 의미한다. Jini 룩업 서비스는 프로그램이 서비스를 검색하는 주 수단이며 djinn(Jini 기반구조에 참가하는 리소스 디바이스, 사용자들의 그룹)상에서 사용자와 관리자가 서비스 발견 및 교신을 위한 서비스를 수행한다.

• Jini Client/Service 프로시저

Jini 아키텍쳐를 이용하기 위해 기존 분산컴퓨팅과 상이한 구조를 사용하여 서비스 등록 및 검색.사용방법을 수행한다. Jini 의 Client/Service 프로세스는 일반적으로 다음과 같다: (a) 서비스는 디스커버리 프로토콜에 의해 룩업 서비스를 발견한다. (b) 서비스는 그 자신을 룩업 서비스에 등록한다. (c) 클라이언트는 룩업 서비스를 발견한다. (d) 클라이언트는 룩업서비스로부터 특정 패러미터를 가지고 서비스를 룩업한다. (e) 클라이언트는 서비스 프록시를 다운로드하여 서비스를 이용한다.[그림 1]



[그림 1] Jini 클라이언트/서비스 프로세스

• Jini 서비스 선택 기법

Jini 라이브러리를 사용하여 클라이언트가 서비스를 선택하는 방법은 다음과 같은 문제점을 안고 있다. 첫째, 투명하고 변환가능한 서비스 선택기법의 어려움이다. 서비스 선택을 위해 [그림 1]의 클라이언트 프로시저에서 2 번을 수행할 때 에트리뷰트(Attribute)를 지정하는 것에 의해 사용자가 요구하는 서비스를 룩업 서비스가 반환한다. 서비스 선택을 위한 코드가 클라이언트에 통합되어 있기 때문에 차후 서비스에 대한 선택 정책의 변화 시에는 클라이언트 코드를 수정해야만 한다. 이러한 코드 관리는 용이하지 않으며 에러를 발생키 쉽다. 둘째, 네트워크 혼잡 가중 및 시스템 성능저하를 가져온다. 일반적으로 클라이언트 서비스 선택을 위해 사용이 쉽고 여러 기능을 제공하는 새로운 클래스인 *ServiceDiscoveryManager* 클래스를 이용하여 서비스 선택을 수행할 수 있다[3]. *ServiceDiscoveryManager* 의 사용은 서비스 선택 시 모든 서비스 리플리카들이 클라이언트 측으로 전송된 후 필터링 되기 때문에 클라이언트 메모리 리소스 사용이 많으며 네트워크 혼잡 또한 초래할 수 있다.

• Jini 인터셉터

인터셉터란 클라이언트나 서버의 호출경로상에 위치하여 메시지나 사용자요구를 중간에 가로채서 적절한 연산을 투명하게 수행하는 기능을 제공해주는 모듈이다. 인터셉터는 보안, QoS, 모니터링, 시각화 등의 다양한 분야에서 사용되어 왔다. 인터셉터의 사용을 통해 Jini 서비스 선택 기법의 문제점을 해결할 수 있다. 현재까지는 Jini 아키텍쳐에 인터셉터와 유사한 개념적 구조가 존재하지 않는다.

본 논문에서는 Jini 인터셉터인 JINT 의 설계 및 구현방안을 제시한다. 기존에 제시된 Jini 부하분산 방안에 JINT 를 적용하는 방식도 제시한다. 논문의 구성은 다음과 같다. 2 절에서는 가능한 두 가지의 JINT 구조 및 설계를 제시한다. 3 절에서는 기존에 제시된 Jini 부하분산방안 및 그 문제점을 기술한다. 4 절에서는 JINT 를 이용한 부하분산방식을 소개한다. 5 절에서는 결론 및 향후 연구과제에 대해 기술한다.

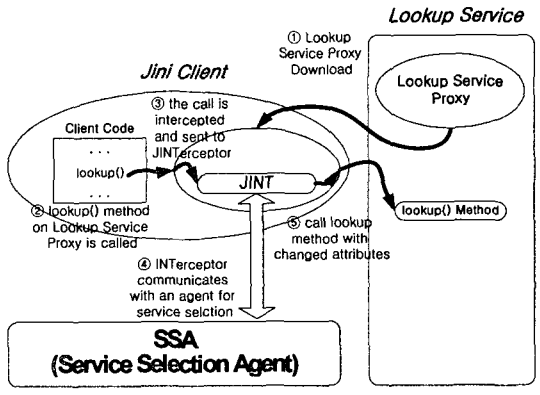
2. JINT : 지니 인터셉터

본 절에서는 Jini 인터셉터인 JINT(JINi INTerceptor)의 설계 및 작동방식을 제시한다. CORBA 인터셉터는 두 가지 종류가 있다[4]. 그러한 인터셉터가 ORB 마다 다른 방식으로 구현될 수 있으며 그 사용 용도도 다를 수 있다[5, 6]. 본 논문에서는 두 가지 JINT 를 제시한다. 룩업 서비스 프록시에 첨가되어 클라이언트 나 서비스 측에서 구동 되어지는 SP JINT 와 룩업 서비스에 첨가되어 서비스 선택 시 호출되는 LS JINT 의 두 가지 JINT 를 제시한다.

JINT 의 주 목적은 서비스 선택을 수행해주는 SSA(Service Selection Agent)를 실행 시에 호출경로 상에 위치시켜 서비스 선택이 특정 정책에 따라 수행되도록 해주는 데 있다. SSA 란 서비스 선택을 위해 클라이언트가 호출한 *lookup* 메소드 호출 파라미터의 하나인 에트리뷰트의 수정, 삭제, 추가 등을 통해 사용자 성향이나 환경에 적절한 서비스를 선택해줄 수 있도록 하는 서비스 선택 대행자이다. SSA 를 연결해주는 JINT 의 사용은 선택 정책의 변화에 용이하게 적응할 수 있는 구조를 제공해 주며 기존 서비스 선택 방식의 주 문제점인 시스템 성능저하 및 네트워크 과부하의 문제점을 해결해 줄 수 있다.

2.1 SP JINT(룩업 서비스 프록시 내장형 JINT)

SP JINT 는 룩업 서비스의 프록시에 내장되어 클라이언트에게 전송된 후 작동된다. 룩업 서비스가 클라이언트 요청에 의해 룩업 서비스 프록시를 전송하는 것은 서비스 사용을 위해 필수적이다. 전송된 룩업 서비스의 프록시를 이용하여 클라이언트는 *lookup* 메소드를 호출하여 원하는 서비스를 검색한다[그림 1]. 일반적인 *lookup* 메소드의 호출은 원격으로 룩업서비스의 *lookup* 을 호출하게 된다.



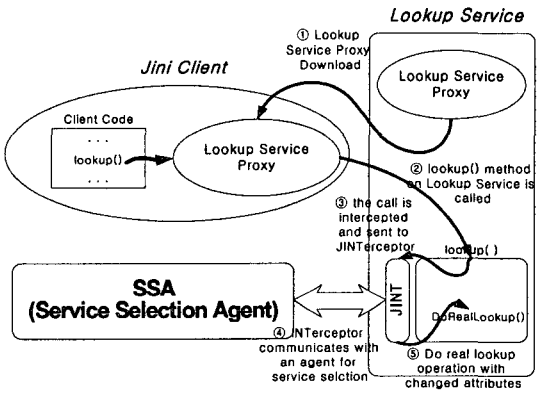
[그림 1] SP JINT : 룩업 서비스 프록시 내장형 JINT

SP JINT 가 탑재된 룩업 서비스 프록시의 경우는 클라이언트의 *lookup* 메소드 호출시 사용되는 에트리뷰트를 SSA 를 통하여 얻어진 속성으로 변경한 후 룩업 서비스에게 *lookup* 메소드를 호출하게 된다. 이러한 동작은 클라이언트 코드에게 투명하게 적용되며 클라이

언트 실행 시 패러미터로서 SSA 를 주는 것에 의해 SSA를 지정할 수 있다.

2.2 LS JINT(룩업 서비스 내장형 JINT)

LS JINT 는 룩업 서비스에 내장되어 서비스 선택을 변경하는 역할을 담당한다. 클라이언트의 *lookup* 메소드 호출은 그대로 룩업 서비스에 전송되고 실제 룩업 서비스 내의 *lookup* 메소드 도중에 삽입된 JINT 에 의해 SSA 와 통신을 한 후 적절한 에트리뷰트를 선정해서 클라이언트로부터 전송된 에트리뷰트를 변경하여 실제 룩업 연산을 수행하도록 한다[그림 2].



[그림 2] LS JINT: 룩업 서비스 내장형 JINT

LS JINT 와 통신을 하여 서비스 에트리뷰트를 반환해주는 SSA 는 룩업 서비스 구동시에 인자로 주어지거나 *Administrable* 인터페이스를 이용하여 변경 및 지정이 가능하다.

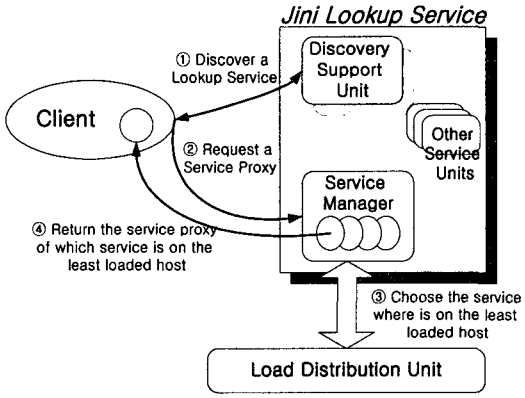
상기에 제시된 SP JINT 와 LS JINT 는 서비스 선택을 위해 동일한 사용 목적을 갖지만 차이점을 가진다. SP JINT 의 경우는 클라이언트 측에서 SSA 와 교신을 하므로 LS JINT 에 비해 서비스 선택에 대한 부하가 분산되는 장점이 있다. 그러나, 클라이언트들이 동일한 SSA 를 사용하는 것이 보장되지 못한다면 서비스 선택이 일관적이게 수행되지 못할 수 있는 단점을 가진다. 이에 비해, LS JINT 의 경우는 룩업 서비스가 SSA 하고 통신을 수행하므로 부하가 집중될 수 있다. 그러나 서비스 선택을 위한 SSA 의 일관적인 사용이 가능하다.

3. 기존 JINI 부하분산 기법

우리는 Jini 부하분산 기법을 [7]에서 제시했다. 제시된 Jini 부하분산 기법은 다음과 같다.

• 룩업 서비스내 부하분산 모듈 통합 기법

이 방식은 룩업 서비스내에서 부하를 분산한다. 클라이언트가 서비스를 요구할 때, 룩업 서비스는 최소 부하를 가진 호스트 상에 존재하는 서비스에 대한 서비스 프록시를 반환한다. 프로시져는 다음과 같다.(a) 클라이언트가 Jini 룩업 서비스를 찾고 난 후 서비스 요청을 한다 (b) 룩업 서비스는 등록된 서비스의 호스트 중 가장 부하가 적은 호스트의 서비스를 찾는다. (c) 룩업 서비스는 서비스 프록시를 반환하고 클라이언트는 서비스를 이용한다[그림 3].

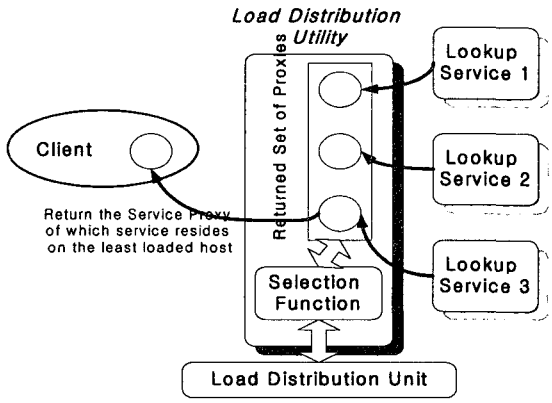


[그림 3] 룩업 서비스내 부하분산 모듈 통합 기법

이 방식은 다음과 같은 장.단점을 지닌다. 클라이언트 및 서비스의 코드가 수정될 필요가 없다. 모든 부하분산 작업이 룩업 서비스 내에서 발생한다. 이 방식의 단점은 한 도메인 내의 모든 룩업 서비스가 이러한 기능을 제공해야 한다는데 있다. 이것은 호환성의 문제를 초래할 수 있다.

• 부하 분산 라이브러리를 통한 부하분산 기법

이 방식은 룩업 서비스를 수정하지 않고 클라이언트 및 서비스가 부하분산을 위한 유틸리티 라이브러리를 사용하여 부하분산을 보장한다. 시나리오는 다음과 같다: (a) 룩업 서비스로부터 중복된 서비스들을 얻는다. (b) 부하분산 장치와 통신을 통해 얻어진 서비스 중 최소 부하를 갖는 호스트상에 상주하는 서비스를 선정하여 클라이언트에게 반환한다[그림 4].



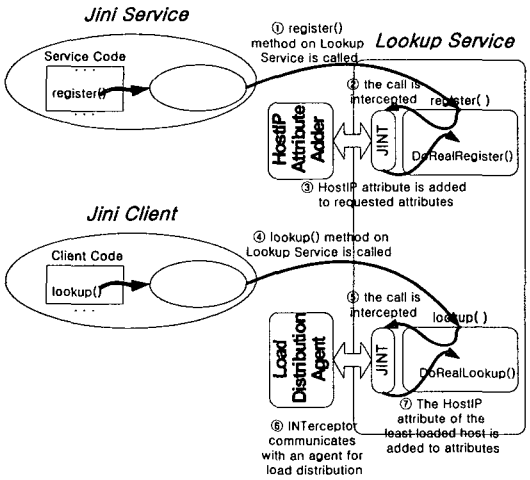
[그림 4] 부하분산 라이브러리를 통한 부하분산 기법

이 방식은 다음과 같은 특성을 지닌다. 첫째, 부하분산을 위한 룩업 서비스의 수정이 필요없다. 둘째, 부하분산 유틸리티를 이용하기 위해 클라이언트 및 서비스의 코드 수정이 요구된다. 또한, 클라이언트 측으로의 많은 수의 서비스 프록시 다운로드는 네트워크 혼잡 및 클라이언트 과부하를 초래할 수 있다.

위에서 기술된 두 가지 부하분산 기법들은 각각의 단점을 지니고 있다. 전자는 룩업 서비스의 호환성을 침해하며 부하분산 기능이 룩업 서비스에 집중된다. 후자는 클라이언트 및 서비스 코드의 변화를 초래한다. 이러한 단점을 극복하기 위해 JINT 를 이용한 Jini 부하분산 기법을 다음절에서 제시한다.

4. JINT를 이용한 Jini 부하분산 기법

본 절에서는 두 가지 JINT 중 LS JINT 를 이용하여 부하분산을 구현하기 위한 방안을 기술한다. LS JINT 는 룩업 서비스의 메소드 중에서 두 가지 함수의 호출에 대해 관여하기 위해 삽입된다. 첫째는 서비스 등록 시 애트리뷰트의 추가를 위해 **register** 메소드의 호출상에 삽입된다. 둘째는 클라이언트로부터의 **lookup** 메소드가 호출되었을 경우 애트리뷰트의 변경, 추가등을 해주기 위해 룩업 서비스의 **lookup** 메소드내에 삽입된다. JINT 를 이용한 부하분산 구조는 [그림 5]과 같다.



[그림 5] LS JINT를 이용한 Jini 부하분산

상기 구조 중에서 HostIP Attribute Adder 는 단순히 등록을 원하는 서비스의 IP 주소를 에트리뷰트 클래스로 변환하여 반환해주는 역할을 하며 Load Distribution Agent 는 CORBA 를 이용하여 작성되었으며 부하분산 장치와 통신을 하여 가장 부하가 적은 호스트의 IP 주소를 얻은 후 그 에트리뷰트를 클라이언트 lookup 호출 인자에 추가하여 요청할 수 있도록 해준다. 이러한 HostIP 및 부하분산 장치에 대한 설명은 [6, 7]에서 기술되었다.

5. 결론 및 향후과제

Jini 는 분산 컴퓨팅을 위한 새로운 아키텍쳐이다. 리싱등을 포함한 다양한 분산 컴퓨팅을 위한 패러다임들이 Jini 에서 제시되었다. Jini 서비스를 이용하기 위해 ServiceDiscoveryManager 와 같은 Jini 라이브러리를 사용하여 서비스를 검색한다. 이러한 서비스 검색을 위한 라이브러리의 사용은 불투명한 서비스 선택 정책 및 시스템 성능저하의 문제점을 가지고 있다.

본 논문은 이러한 문제점들을 해결하기 위해 Jini 를 위한 인터셉터인 JINT 의 설계 및 구현방안을 제시하였다. JINT 의 사용은 투명한 서비스 선택을 가능케 해주며 네트워크 혼잡 및 과대한 시스템 리소스를 사용하는 서비스 프록시 다운로드의 문제점을 해결할 수 있다. 본 논문에서는 룩업 서비스 프록시 내장형의 SP JINT와 룩업 서비스 내장형인 LS JINT의 두 가지 JINT 를 제시했다. 마지막으로 기존에 제시된 Jini 부하분산 기법들에 대해 언급하고 그 기법들이 가지고 있는 단점을 해결하기 위해 LS JINT를 이용한 부하분산 방법을 제시하였다.

JINT 는 부하분산 이외에도 시각화, 로깅등의 여러 다른 응용분야에도 사용 가능하다. 다양한 분야에 사용되기 위해 확장 가능한 JINT 의 설계 및 구현이 요구된다.

참고문헌

[1] Sun Microsystems, Inc., "Jini Architectural Overview," http://www.sun.com/jini/whitepapers/-architecture.html

[2] J. Waldo, "The Jini Architecture for Network Centric Computing", ACM Computing Magazine

[3] K. Edwards, " Core Jini 2nd Edition," Prentice Hall, 2000.

[4] Object Management Group. The Common Object Request Broker: Architecture and Specification. Revision 2.3, Object Management Group, Framingham, Mass., June 1999.

[5] Priya Narasimhan, Louise E. Moser, and P.M. Melliar-Smith, Using Interceptors to Enhance CORBA, Computer Magazine Vol. 32, p.62-68. No. 7, July 1999

[6] Kim, Joong-Han, et al. "LODIN : Load Distribution of CORBA Object Using Interceptor," IEEE TENCON 2001, Accepted.

[7] Kim, Joong-Han, et al. " Load Distribution Strategies in Jini," KISS, April 2001, Submitted.