SecureJS : Jini2.0 기반의 안전한 JavaSpace

유 양 우[†] · 문 남 두[††] · 정 혜 영[†††] · 이 명 준[†††]

요 약

Jini 서비스는 개발자에게 분산시스템을 쉽게 개발할 수 있는 하부구조를 제공한다. Jini 서비스 중 하나인 *JavaSpace*는 자바환경의 분산 컴퓨팅 모델로서 객체를 저장하고 저장된 객체에 접근할 수 있는 공간을 말한다. 이러한 JavaSpace 서비스는 객체를 공유하는 방법으로 매우 유용하게 사용되고 있지만, 보안성이 취약하여 객체정보에 대한 접근 보안이 요구되는 분산시스템의 개발에는 적합하지 않다. 본 논문에서는 JavaSpace의 취약한 보안성을 강화시켜 안전한 JavaSpace 서비스를 제공하는 *SecureJS* 시스템에 대하여 설명한다. *Jini2.0* 기반의 SecureJS 시스템은 자바객체를 저장할 수 있는 ObjectStore와 사용자에 대한 ObjectStore의 접근을 제어하는 AccessManager 그리고 공개키를 관리하는 KeyManager로 구성되어 있다.

SecureJS : A Secure JavaSpace based on Jini2.0

Yang-Woo Yu[†] · Nam-Doo Moon[††] · Hye-Young Jung[†††] · Myung-Joon Lee[†††]

ABSTRACT

The Jini system provides an infrastructure to facilitate a programmer to develop distributed systems. As one of the Jini services, *JavaSpace* has been used as a repository which is accessible publicly in the Java distributed environment. Although JavaSpace could give a useful method for saving and sharing java objects, it would not be applicable to develop a distributed system requiring access securities for the objects because JavaSpace does not support secure access control. In this paper, we present a secure JavaSpace service based on Jini2.0 named *SecureJS*, which strengthens the security weakness of JavaSpace. The system consists of ObjectStore to store Java objects, AccessManager to control access of ObjectStore and KeyManager to manage public keys.

키워드 : Jini2.0, 자바스페이스(JavaSpace), SecureJS, LDAP, 분산시스템(Distributed System)

1. 서 론

인터넷 기술은 빠르게 발전하고 있으며 그에 따른 활용할 수 있는 분야도 더욱더 다양해졌다. 네트워크의 개념 또한 다수의 연결된 컴퓨터에서 유용한 서비스를 제공하는 장치로 그 영역이 점점 더 확대되어 가고 있다. 이러한 네트워크의 발전과 함께 시스템들의 상호협력을 위하여 객체를 공유하고, 동적인 통신을 가능하게 해주는 새로운 분산 기반 구조인 Jini 기술이 등장하였다[1, 2].

Jini 기술은 네트워크 상의 지능형 기기 또는 소프트웨어들이 Jini 네트워크에 접속과 동시에 서비스할 수 있는 기능(Network Plug and Work)을 제공하고, 사용자가 서비스를 요청하면 요청한 서비스를 찾아 처리를 할 수 있는 새로운 분산 기반 구조를 제공한다[3]. Jini 기반 구조는 서비스를 등록하기 위하여 룩업(Lookup)서비스를 찾는 디스커버리(Discovery) 프로토콜과 등록된 서비스와 연결을 위한 조인(Join) 프로토콜이 있으며, 그리고 클라이언트가 서비스를 찾기 위한 룩업 프로토콜이 있다[4, 5]. 또한 Jini 서비스로서 분산 응용프로그램 개발을 위하여 룩업, 리스, 트랜잭션 그리고 JavaSpace 서비스를 제공하고 있다.

Jini 서비스 중에 하나인 JavaSpace는 표준 인터페이스를 통하여 객체를 저장하고, 저장된 객체를 그 클래스의 템플릿을 이용하여 읽거나 가져오는 새로운 패러다임을 갖고 있다[6]. 이러한 특징을 이용하여 분산객체의 영속성과 데이터 교환 기능을 지원하는 분산시스템을 구현하는데 많이 활용되고 있다. 하지만, 기존의 JavaSpace는 보안기능이 취약하여 누구나 객체를 저장하고 저장된 객체를 아무런 제약 없이 누구든지 접근할 수 있었다. 그러므로 보안이 요구되는 정보를 교환하거나 보관할 경우 JavaSpace는 객체 저장소로서의 서비스를 제공할 수 없는 단점을 가지고 있다.

최근에 썬 마이크로시스템즈(Sun Microsystems)에서는 분산시스템 환경에서 자바코드의 이동성에 따른 보안요소를 충족시키는 Jini2.0을 제시하였다[1, 7]. 본 논문에서는 새로운 Jini2.0 보안모델을 적용하여 서비스에 대한 상호인증과 객체

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.
교신저자(이명준 : mjlee@ulsan.ac.kr)
† 준 회 원 : 울산과학대학 컴퓨터정보학부 교수
†† 준 회 원 : 울산대학교 대학원 컴퓨터정보통신공학부
††† 정 회 원 : 울산대학교 컴퓨터정보통신공학부 교수
논문접수 : 2004년 6월 19일, 심사완료 : 2004년 8월 4일

에 대한 접근제어 기능을 구현하였으며, JavaSpace 서비스를 안전하게 접근할 수 있도록 SecureJS 시스템을 개발하였다.

본 논문의 구성은 다음과 같다. 2장에서는 JavaSpace 서비스의 소개하고, 새롭게 추가된 Jini2.0의 프로그래밍 모델과 보안모델에 관하여 서술한다. 3장에서는 안전한 JavaSpace 서비스를 제공하기 위하여 Jini2.0 기반의 SecureJS 시스템의 설계 및 구현에 관하여 자세히 설명한다. 그리고, 4장에서는 SecureJS 시스템을 이용한 활용예제를 설명하며 시스템의 동작방식과 운영에 관하여 살펴본다. 끝으로 5장에서 결론 및 향후 연구방향에 대해 살펴본다.

2. 관련 연구

2.1 JavaSpace 서비스의 소개

JavaSpaces™ 기술은 자바 환경의 새로운 분산 컴퓨팅 모델로서 자바객체를 저장하고 접근할 수 있는 공간을 말한다. JavaSpace는 1980년대 Yale 대학에서 개발한 분산 애플리케이션 개발 툴인 Linda에서 영향을 받았으며, Linda는 네트워크 시스템으로 연결된 환경에서 병렬 분산 처리를 용이하게 구현할 수 있음을 보여주었다[8].

JavaSpace는 자바의 원격 객체 호출 시스템인 RMI(Remote Method Invocation)와 직렬화를 이용하며, 룩업(lookup) 서비스, 트랜잭션(transaction) 서비스 등 Jini 서비스와 연계하여 분산처리를 위한 기능들을 제공함으로서 분산 시스템을 쉽게 구축할 수 있도록 한다.

기존의 분산 애플리케이션이 프로세스 간 통신 시 직접 메시지를 전달하거나 원격 객체의 메소드를 호출하는 방식으로 이루어지는 반면 JavaSpace를 이용한 통신 모델에서는 프로세스들이 직접 통신하지 않고 JavaSpace라는 분산 데이터 저장소를 통해 객체를 교환하게 된다[6, 9].

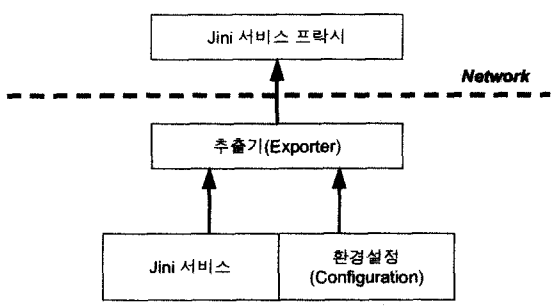
2.2 JavaSpace 서비스에서 보안요소의 필요성

JavaSpace를 이용한 프로그래밍 모델은 매우 간결하다. JavaSpace를 이용하고자 하는 응용프로그램은 Jini의 룩업(Lookup) 서비스를 이용하여 JavaSpace에 접근할 수 있는 서비스프락시를 다운로드 한다. 그런 다음 서비스프락시를 이용하여 객체를 저장하고, 원하는 객체를 검색하여 그 객체를 읽거나 가져갈 수 있다. 이러한 패러다임은 객체를 공유하는 차원에서는 매우 유용하게 사용될 수 있다. 그러나 JavaSpace의 프락시를 룩업 서비스로부터 구하기만 하면 어떠한 목적의 응용프로그램도 JavaSpace 내의 공간에 객체를 저장하거나 가져갈 수 있기 때문에 이러한 객체정보들에 대한 접근 보안이 요구되는 분산 응용프로그램 개발에는 적합하지 않다.

따라서, 본 연구에서는 기존의 JavaSpace의 기능에 Jini2.0 보안요소를[4, 7] 추가하여 안전한 JavaSpace 서비스를 제공하고자 한다. 안전한 JavaSpace 서비스를 제공하기 위해 다음의 두 가지 사항을 고려한다. 첫째는 신원이 허용되고 확인된 클라이언트만이 룩업 서비스로부터 JavaSpace의 서비스프락시를 구할 수 있도록 제한한다. 둘째는 JavaSpace에 객체를 저장할 때 객체에 접근할 수 있는 수신자를 지정하고, 지정된 수신자만이 객체에 접근할 수 있도록 제한한다. 서비스프락시에 대한 접근을 제한하기 위해 Jini2.0 기반의 JERI 모델과 JAAS 인증 기능을 이용하였다[10]. 그리고 JavaSpace 내의 객체에 대한 접근을 제한하기 위하여 *SecureJS 시스템*을 개발하였다. SecureJS 시스템은 AccessManager와 ObjectStore로 구성되어 있으며, 시스템에 관한 자세한 설명은 3장에서 논의할 것이다.

2.3 Jini2.0의 소개

Jini2.0 시스템 구조는 프로그래밍 모델, 하부구조 그리고 서비스들로 구성되어 있다. 이전 버전의 Jini 서비스는 룩업 서비스, 트랜잭션 관리, 리스, 이벤트처리, JavaSpace 서비스 등을 제공하였으며, Jini2.0은 이러한 서비스의 지원과 더불어 추가적으로 새로운 *프로그래밍 모델*과 *하부구조*를 추가시켰다[3, 4]. 새롭게 추가된 프로그래밍 모델에는 환경설정(Configuration), 추출기(Exporter), 프락시 준비기(Proxy-Preparer)를 지원하며, 하부구조는 보안과 호출제약(Invocation Constraints) 그리고 JERI 등이 추가되었다[7]. Jini2.0의 확장된 기능의 지원으로 기존의 보안성이 취약한 Jini 기반의 응용프로그램들에 대하여 강력한 보안요소와 보안에 맞는 다양한 환경설정을 제공하고, 클라이언트 측과 서버 측의 프로그래밍 모델을 단일화할 수 있게 되었다. 또한 JERI를 지원함으로써 다양한 전송계층을 사용할 수 있다는 큰 장점을 갖고 있다. (그림 1)은 Jini2.0의 새로운 프로그래밍 모델을 간략히 표현하고 있다.



(그림 1) Jini2.0 프로그래밍 모델

Jini 기반의 응용 프로그램에서 클라이언트는 서비스를 받기 위해 룩업서비스를 이용하여 서비스프록시를 다운로드 받아야 한다. 그리고, 이 서비스프락시 내의 메소드를 호출함으로 써 원하는 서비스를 이용할 수 있다. 대다수 응용프로그램들은 안전한 원격 통신을 위하여 다음과 같은 보안사항을 요구한다.

- 클라이언트와 서버 사이의 상호인증
- 액세스 제어를 위한 권한 제한

- 무결성
- 암호화를 통한 기밀성

이러한 요구사항들은 안전한 데이터통신 위하여 필요한 전형적인 보안 요구사항이다[7]. 기존의 Jini1.0에서는 이러한 보안 요구사항을 지원하지 않았으며 보안은 개발자의 영역으로 남겨두었었다. 하지만, Jini2.0 모델이 새롭게 출시되면서 위에서 제시한 보안 요구사항을 모두 지원하며 추가적으로 코드무결성(code integrity)을 지원하고 있다.

Jini와 자바 RMI 모델에서, 객체들의 데이터는 원격호출로 전송되지만 그에 대한 코드 즉 서비스프락시는 원격호출과는 달리 다운로드 해야한다. 그래서, 응용프로그램은 받은 객체 즉, 객체코드와 데이터들에 대하여 반드시 "trust"임을 검증해야한다. Jini2.0 보안모델에서는 응용프로그램이 원격 호출 시, 다양한 요구사항을 명시할 수 있는 제약(constraints) 기반의 원격호출 모델을 지원한다. 이러한 모델은 코드무결성을 보장하는데 사용되는 메커니즘으로 제공된다.

2.3.1 제약(Constraints) 기반의 원격호출

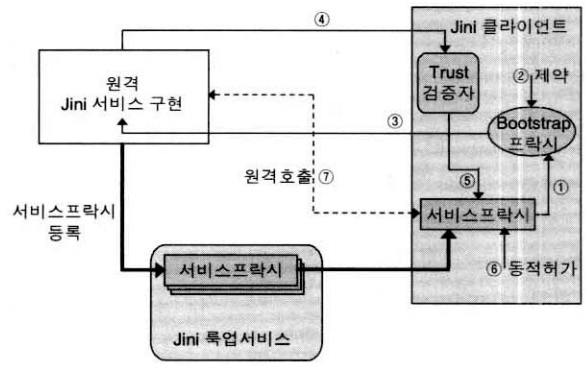
위에서 제시한 보안 요구사항들은 원격호출 시 제약(constraints)으로 표현된다. 제약은 서비스프록시를 이용하여 서비스와 원격 호출 시, 클라이언트가 호출하는 방법에 대하여 설정할 수 있는 특정 요구사항이다. 프락시는 클라이언트들이 제약을 설정할 수 있도록 net.jini.core.constraint.RemoteMethodControl 인터페이스를 구현한다. 프락시의 역할은 이러한 제약을 만족시키고, 클라이언트는 그 프락시가 올바른 프락시인지를 검증하는 것이다. 각 제약은 net.jini.core.constraint.InvocationConstraint의 인스턴스로 표현된다. 프락시는 클라이언트에서 설정한 제약들을 만족시키기 위하여 적절한 전송방법과 프로토콜을 사용해야 한다. Jini2.0에서 보안구현은 각 계층별로 필요에 따라 서로 다른 전송방법과 프로토콜을 플러그인(plug-in) 할 수 있으며, SSL과 Kerberos와 같은 폭넓은 프로토콜을 지원한다[10]. 본 연구에서는 SSL 프로토콜을 사용하였다.

2.3.2 프락시 준비(Proxy Preparation)

클라이언트는 프락시에 대한 제약들을 설정하기 이전에, 설정된 제약들을 실행하는 프락시를 신뢰할 수 있는지 여부를 확인할 필요가 있다. 신뢰할 수 없는 곳에서 프락시를 받았을 경우, 클라이언트는 프락시의 메소드를 호출하기 이전에 다음과 같은 방법으로 프락시를 준비(prepare)하게 된다.

① 다운로드 받은 서비스프락시를 검증하기 위하여 Bootstrap 프락시를 호출한다.
② Bootstrap 프락시는 클라이언트로부터 서비스와 통신에 따른 제약을 설정한다.
③, ④ 클라이언트는 서비스객체를 호출하여 "Trust 검증자"를 다운로드한다.
⑤ Trust 검증자를 통하여 서비스프락시를 검증한다.
⑥ 검증된 서비스프락시에게 허가를 부여한다.
⑦ 서비스프락시에 대한 새로운 제약을 설정하고, 서비스객체와 통신한다.

위의 단계들은 Jini2.0에서 프락시 준비(Proxy preparation)로 알려진 하나의 오퍼레이션 prepareProxy()로 캡슐화 되어있다.



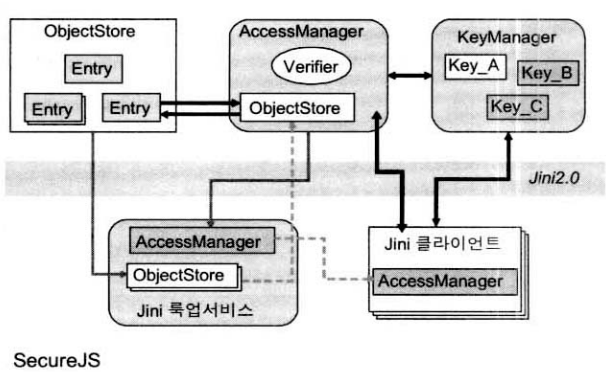
(그림 2) 프락시에 대한 신뢰성 검증

3. SecureJS 시스템

JavaSpace는 분산 객체의 공유개념을 도입하여 누구든지 자바객체를 저장하고, 저장된 객체에 접근하여 읽거나 가져갈 수 있는 Jini 서비스이다. 기존의 JavaSpace는 보안성을 제공하지 않으므로 본 연구에서는 Jini2.0 보안모델과 프로그래밍 모델을 기반으로 보안성이 강화된 JavaSpace 서비스를 제공하는 *SecureJS 시스템*을 개발하였다.

3.1 SecureJS 시스템의 보안정책

SecureJS 시스템은 *AccessManager*(접근관리자)와 *ObjectStore*(객체저장소) 그리고 *KeyManager*(키 관리자)로 구성되어 있다. ObjectStore는 JavaSpace 서비스를 제공하는 객체저장소이며, AccessManager는 인증된 클라이언트들에게 안전한 JavaSpace 서비스를 제공하는 데몬 형태로 동작하는 Jini2.0 서비스이다. 그리고, KeyManager는 Access



(그림 3) SecureJS 시스템의 구조

Manager에서 올바른 수신자인지 여부를 검사할 때 사용하는 공개키들을 관리한다[11]. 이러한 구성요소를 갖는 SecureJS 시스템은 분산 컴퓨팅 환경에서 객체를 안전하게 공유하고 저장할 수 있는 매우 유용한 기술을 제공하고 있다. (그림 3)은 SecureJS의 구조를 보여주고 있다.

3.1.1 ObjectStore(객체저장소)

JavaSpace는 엔트리(net.jini.core.entry.Entry) 인터페이스를 구현한 자바객체를 저장하는 Jini 서비스이다. 엔트리는 java.io.Serializable을 확장한 인터페이스이며, 엔트리 객체의 모든 속성은 직렬화할 수 있는 객체 참조이어야 한다. 사용자는 JavaSpace에 엔트리를 구현한 객체를 저장할 수 있으며, 엔트리 객체의 템플릿을 이용하여 매칭되는 엔트리 객체를 구할 수 있다.

ObjectStore는 하나의 JavaSpace이며 Jini2.0 보안정책을 이용하여 JavaSpace에 접근할 수 있는 모든 권한을 AccessManager에게만 부여한다. 이러한 작업을 위해 ObjectStore는 Jini2.0 환경설정 파일을 작성해야 하며 (그림 4)에서 ①, ② 코드는 JavaSpace의 접근권한을 AccessManager에게만 부여하는 환경설정의 일부분이다.

```
Server.ObjectStore {
   private static AccessManagerKey = KeyStores.getX500Principal
                ("AccessManager",caTruststore) ; -------- ①
                :
   AuthenticationPermission(AccessManagerKey, ObjectStoreKey,
                "connect") ;                    -------- ②
}
```

(그림 4) ObjectStore의 환경설정

ObjectStore에 저장되는 자바객체는 엔트리와 수신자 id 정보를 포함한다. 수신자 id는 저장된 엔트리에 대하여 올바른 수신자에게 정확히 요청한 객체를 검색하여 전달하기 위해 사용된다. (그림 5)에 기술된 코드는 SecureJS 시스템에서 사용하는 엔트리의 구조를 보여준다.

```
import net.jini.core.entry.Entry ;
public class Id_Entry implements Entry {
    private String receiver_id ;
    private Entry entry ;

    public Id_Entry(String rcv_id, Entry e) {
      receiver_id = rcv_id ;
      entry = e ;
    }
    public String get_id() {
      return receiver_id ;
    }
    public Entry get_entry() {
      return entry ;
    }
}
```

(그림 5) ObjectStore의 환경설정

3.1.2 AccessManager(접근관리자)

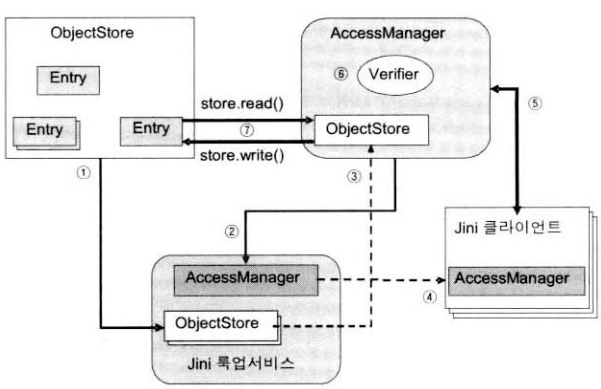
일반적으로 클라이언트들은 JavaSpace 서비스를 이용하기 위하여 Jini 룩업서비스로부터 서비스프락시를 다운로드 받는다. 기존의 Jini 모델에서는 인증과정 없이 서비스프락시를 요구하는 모든 클라이언트들에게 이러한 프락시를 제공함으로써 서비스에 대한 보안성이 매우 부족하였다.

SecureJS 시스템은 JavaSpace 서비스에 대한 두 가지 보안사항을 적용하였다.

- JavaSpace에 접근할 수 있는 프로세스를 AccessManager로 한정시켜 클라이언트들 이 JavaSpace에 접근할 경우 AccessManager를 통해서만 접근 할 수 있다.
- JavaSpace에 저장된 엔트리에 대하여 클라이언트들이 접근하고자 할 때, AccessManager는 적법한 수신자임을 검증하여 그 클라이언트에게 서비스를 제공한다.

SecureJS 시스템은 자바객체를 저장할 수 있는 공간으로 ObjectStore를 사용하고 있으며, ObjectStore에 대한 접근은 AccessManager만이 권한을 부여받아 서비스를 제공할 수 있다. 이는 (그림 6)의 환경설정 파일에서 지정한다. ObjectStore에 접근할 수 있는 인증된 주체를 "AccessManager"로만 한정시켰다. 그래서, 클라이언트들은 안전한 JavaSpace의 서비스 이용하고자할 때, AccessManager를 통해서만 서비스 받을 수 있다. 즉, 클라이언트들은 ObjectStore 서비스에 직접 접근 할 수 없다. 이러한 설계는 클라이언트들에게 안전한 JavaSpace 서비스를 제공하기 위한 새로운 보안모델로서 제시된다.

두 번째 보안사항은 JavaSpace에 저장된 엔트리에 대한 보안으로 인증된 클라이언트라 할지라도 그 엔트리를 접근할 수 있는 수신자가 아니라면 접근할 수 없도록 한 것이다. 이는 **[올바른 수신자의 검증 알고리즘]**에서 자세히 설명할 것이다. (그림 6)은 AccessManager의 동작을 설명하고 있다.



(그림 6) AccessManager의 동작

① 자바객체를 저장하는 ObjectStore는 Jini 서비스로 동작하기 위해 룩업서비스에 자신의 서비스프락시를 등록시킨다. 그리고 자신을 접근할 수 있는 유일한 접근자로서

AccessManager를 지정한다.

② AccessManager 또한 클라이언트들이 자신의 서비스를 받을 수 있도록 룩업서비스에 서비스프락시를 등록시킨다.

③ AccessManager는 Jini2.0 보안모델을 이용하여 ObjectStore의 프락시를 다운로드 한다. AccessManager를 제외한 다른 클라이언트들은 ObjectStore의 프락시를 다운로드 받을 수 없다.

④ 안전한 JavaSpace 서비스를 이용하기 위하여, 먼저 클라이언트들은 인증과정을 수행한 후, 룩업서비스를 통하여 AccessManager의 서비스프락시를 다운로드 받는다.

⑤ 클라이언트들은 다운로드 받은 AccessManager의 프락시를 이용하여 원격메소드를 호출한다.

⑥ AccessManager의 검증자(verifier)는 허가된 접근인지 여부를 검사한 후 적법한 클라이언트임이 검증되면 ObjectStore의 서비스프락시 내에 메소드를 호출하여 안전한 JavaSpace 서비스를 제공한다.

⑦ ObjectStore는 AccessManager에서 요구한 JavaSpace 서비스를 수행한다. 그리고 그 결과 값을 클라이언트에게 반환한다.

SecureJS 시스템은 기존의 JavaSpace에서 제공하는 오퍼레이션을 모두 지원하고 있으며, 클라이언트들은 정의된 오퍼레이션을 수행함으로써 안전한 JavaSpace 서비스를 제공받을 수 있다. SecureJS 시스템의 주요 오퍼레이션들은 AccessManager_Impl 클래스 내에 정의되어 있으며, 이를 이용하여 엔트리를 저장하고 검증된 수신자가 저장된 엔트리를 읽는 과정을 설명한다.

[올바른 수신자의 검증 알고리즘]

① 인증된 클라이언트는 엔트리(**entry**)를 SecureJS 내에 write() 메소드를 이용하여 저장할 수 있다. 일반적인 JavaSpace의 write() 메소드와는 달리 SecureJS에서는 그 엔트리에 접근할 수 있는 수신자의 id(**receiver_id**)를 write() 메소드에 매개변수로 지정한다. 이는 올바른 수신자를 검증하기 위해서이다.

② 엔트리에 수신자를 지정하여 SecureJS 시스템의 **write (receiver_id, entry, txn, lease)** 메소드를 호출하여 엔트리를 저장한다.

③ 클라이언트는 read() 메소드를 이용하여 ObjectStore 내에 저장된 엔트리에 접근을 시도할 때, AccessManager는 저장된 객체를 서비스하기 전에 그 클라이언트가 해당 객체에 접근할 수 있는 올바른 수신자인지 검증해야한다.

④ 먼저, 수신자는 자신이 올바른 수신자임을 증명하기 위하여 자신의 id를 개인키로 암호화시킨 값과 자신의 id를 read() 메소드를 이용하여 매개변수로 전달한다.
예) **read(private_encrypt_id, id, id_tmpl, txn, lease)**

⑤ AccessManager는 수신자의 개인키로 암호화시킨 값과 수신자 id 그리고 KeyManager를 이용한 수신자의 공개키 정보를 이용하여 올바른 수신자임을 검증할 수 있다.

⑥ KeyManager에서 구한 수신자의 공개키를 사용하여 수신자의 개인키로 암호화된 값을 해독하여 id 값을 구한다. 이 값과 수신자가 보낸 id가 일치하면, 이는 올바른 수신자임이 검증된 것이다.

AccessManager는 AccessManager_Impl 클래스를 이용하여 구현된다. 다음 코드는 AccessManager_Impl 클래스의 write()와 read() 메소드를 이용하여 ObjectStore에 엔트리를 저장하고, 저장된 엔트리에 접근하기 위하여 올바른 수신자임을 검증하는 과정을 보여주고 있다.

클라이언트는 *write*() 메소드를 호출하여 ObjectStore에 엔트리를 저장한다. **receiver_id**인자는 엔트리에 접근할 수 있는 수신자의 id이다. 그리고 **entry** 인자는 ObjectStore 내에 저장하고자 하는 자바객체 이다. (그림 7)은 AccessManager의 write() 메소드를 보여준다.

```
/* 객체를 저장하기 위하여 ObjectStore의 write() 메소드를 호출한다. */
public Lease write(receiver_id, entry, txn, lease) {
    Id_Entry ie = new Id_Entry(receiver_id, entry) ;
    object_store.write(Id_Entry, txn, lease) ;
}
```

(그림 7) AccessManager의 write() 메소드

ObjectStore에 저장된 엔트리에 접근하기 위하여 *read*() 또는 *take*() 메소드를 이용할 수 있다. AccessManager는 클라이언트가 이들 오퍼레이션을 호출할 때, 호출한 클라이언트가 올바른 수신자인지를 검증하고, 검증된 수신자라면 해당 오퍼레이션에 따른 ObjectStore 서비스를 제공한다. (그림 8)은 read() 메소드를 설명하고 있다.

```
/* 저장된 객체를 읽기 위하여 ObjectStore의 read() 메소드를 호출한다. */
public Entry read(private_encrypt_id, id, tmpl, txn, lease) throws
    NotTrusted {
    boolean trust ;

    ldap_pub_key = ldapCtx.search(id, PK_FILTER, constraints) ;
                                                          ---------- ①
    trust = verifier(ldap_pub_key, private_encrypt_id, id) ; ---------- ②

    if (trust) {
      Id_Entry id_tmpl = new Id_Entry(id, tmpl) ;          ---------- ③
      Id_Entry ie = object_store.read(id_tmpl, txn, lease) ; ---------- ④

      return ie.get_enry() ;                               ---------- ⑤
    }
          :
```

(그림 8) AccessManager의 read() 메소드

read() 오퍼레이션에서 첫 번째 인자 **private_encrypt_id**는 올바른 수신자임을 검증하기 위해 수신자 자신의 id를 수신자 개인키(private key)로 암호화시킨 값이다. 그리고,

두 번째 인자는 엔트리에 접근을 허용할 수신자의 **id**이다. **tmpl**인자는 수신자가 접근하고자하는 엔트리를 검색하기 위한 템플릿이다. read() 오퍼레이션의 동작은 다음과 같다.

① LDAP를 이용하는 KeyManager에 접근하여 수신자의 id에 해당하는 수신자의 공개키를 얻어온다[12, 13].

② *verifier*(*ldap_pub_key*, *private_encrypt_id*, *id*) 메소드를 이용하여 수신자가 올바른 수신자인지를 검사한다. *ldap_pub_key*인자는 KeyManager의 search() 오퍼레이션을 통하여 얻은 수신자의 공개키이며, *private_encrypt_id*는 수신자 id를 수신자 자신의 개인키로 암호화시킨 값이다. 그리고 id는 수신자 자신의 id이다.
: AccessManager는 수신자의 공개키를 가지고 수신자의 개인키로 암호화시킨 **private_encrypt_id**를 해독한다. 해독한 값이 read() 메소드의 두 번째 인자인 id값과 일치하면 올바른 수신자임이 검증된 것이다.

③ **id_tmpl** 인자는 ObjectStore에 저장된 엔트리를 검색하기 위한 템플릿이다.

④ 템플릿에 정확히 매칭되는 엔트리를 ObjectStore의 read() 메소드를 호출하여 가져온다.

⑤ 검색된 엔트리에서 실제 수신자가 요구하는 엔트리만을 수신자에게 전달한다.

3.2 KeyManager(키 관리자)를 이용한 공개키 관리

SecureJS 시스템에서 AccessManager는 사용자의 신원을 인증하기 위한 방법으로 JDK에서 지원하는 DSA 보안 알고리즘을 사용하며[10, 11], 각 사용자들에 대한 공개키와 개인키를 구할 수 있다. 그리고 DSA 알고리즘을 이용하여 얻은 공개키는 LDAP를 이용하는 KeyManager에서 관리된다[13]. AccessManager와 클라이언트는 상호인증을 위해 KeyManager에서 관리되는 공개키를 사용한다.

SecureJS 시스템에서 KeyManager는 공개키 관리를 위하여 LDAP 디렉토리를 사용한다. 디렉토리는 검색할 정보의 커다란 집합으로 볼 수 있다. 전화번호부와 같이 정보에 거의 변화가 없고, 매우 자주 검색되는 정보는 디렉토리의 영역이라고 볼 수 있다. LDAP는 국제 표준기구(ISO)와 국제 원거리 통신 연맹(ITU)에서 승인한 디렉토리 표준인 X.500 기술의 문제점을 극복하기 위해 고안되었다. LDAP는 인터넷의 TCP/IP 프로토콜을 사용할 수 있도록 설계되었으며, X.500과 비교하면 자원소모가 적고 기능이 단순화되었다. 네트워크 디렉토리를 특수한 데이터베이스로 생각한다면 LDAP를 이용하여 관련 서버와 서버 주변기기, 어플리케이션이나 문서의 위치를 파악하는데도 많은 도움을 주므로 그 기능을 크게 확장시켜 나갈 수 있다[14]. KeyManager의 주요 기능은 다음과 같다.

- 바인딩(Binding) : LDAP는 바인딩과 인증을 구별하지 않으며, 디렉토리 서버로 바인딩할 때 원하는 서버와 자신에 대한 정보(계정, 암호)를 함께 지정할 수 있다.
- 검색(Searching) : 검색을 하려면 검색의 범위를 지정하여 *search*() 메소드를 호출해야 한다.
- 엔트리 추가(Adding Entries), 엔트리 변경(Modifying Entries), 엔트리 제거(Deleting Entries) : 디렉토리 구조에서 관리되는 각 엔트리를 추가, 변경, 삭제할 수 있다.

3.3 AccessManager 서비스 객체

AccessManager는 데몬 형태로 동작하며 인증된 클라이언트에게 서비스를 제공하고 ObjectStore 서비스를 이용하기 위하여 Jini2.0 기반의 프로그래밍 모델과 보안모델을 따라야 한다. (그림 9)는 AccessManager 클래스의 구현코드를 설명하고 있다.

```
public class AccessManager {
   private AccessManager_Impl       secureJS ;
   public static void main(String[ ] args) throws Exception {

① ----- /* AccessManager.config 파일을 불러온다. */
      final Configuration config = ConfigurationProvider.getInstance(args) ;
      LoginContext loginContext = (LoginContext) config.getEntry(
                  "Server.AccessManager", "loginContext",
                   LoginContext.class, null) ;

② ----- /* 서비스 디스커버리 관리자의 위치정보를 찾는다. */
      ServiceDiscoveryManager serviceDiscovery = (ServiceDiscoveryManager)
                  config.getEntry("Server.AccessManager", "serviceDiscovery",
                                  ServiceDiscoveryManager.class) ;

③ ----- /* AccessManager를 추출시킨다. */
      Exporter exporter = (Exporter) config.getEntry( "Server.AccessManager",
                  "exporter", Exporter.class, new BasicJeriExporter
                  (SslServerEndpoint.getInstance(0), new BasicILFactory())) ;
      serverProxy = (AccessManager) exporter.export(this) ;

④ ----- /* 조인 관리자를 생성하여, AccessManager 서비스를 룩업서비스에 등록시킨다. */
      JoinManager joinManager = new JoinManager(serverProxy, null,
                                    getServiceID(), discoveryManager, null,
                                    config) ;

⑤ ----- /* ObjectStore 서비스의 프락시를 찾는다. */
      ServiceItem serviceItem = serviceDiscovery.lookup(
                  new ServiceTemplate(null, new Class[ ] { JavaSpace.class },
                  null), null, Long.MAX_VALUE) ;
      JavaSpace object_store = (ObjectStore) serviceItem.service ;

⑥ ----- /* ObjectStore의 서비스프락시를 준비(prepare)한다. */
      ProxyPreparer preparer = (ProxyPreparer) config.getEntry
                        ("Server.AccessManager", "preparer", ProxyPreparer.
                         class, new BasicProxyPreparer()) ;
      object_store = (JavaSpace) preparer.prepareProxy(object_store) ;

⑦ ----- /* 엔트리를 저장하기 위하여 ObjectStore의 write() 메소드를 호출한다. */
      object_store.write(id_entry, txn, lease) ;
                  :
                  :
}
```

(그림 9) AccessManager 서비스 객체의 구현

① AccessManager에 대한 환경설정(Configuration) 파일을 getInstance() 메소드를 이용하여 읽어온다.
② AccessManager 서비스를 룩업서비스에 등록하기 위하여 서비스 디스커버리 관리자의 위치정보를 구한다.
③ 인증된 클라이언트들이 접근할 수 있도록 AccessManager의 서비스프락시를 Jini 룩업서비스에 등록시키기 위한 코드이다. 추출기(exporter)를 이용하여 서비스프락시를 작성한다.
④ 작성된 AccessManager의 서비스프락시인 serverProxy를 조인관리자를 통하여 룩업서비스에 등록한다.
⑤ AccessManager는 ObjectStore 서비스를 이용해야 함으로 ObjectStore의 서비스프락시를 구해야 한다. 이는 서비스 디스커버리 관리자의 lookup() 메소드를 호출하여 ObjectStore의 서비스프락시를 찾는다.
⑥ AccessManager는 ObjectStore의 서비스프락시를 받아서 이 프락시가 신뢰할 수 있는 곳에서 다운로드 했는지 여부를 프락시 준비기(ProxyPreparer)를 통하여 검증한다.
⑦ 검증과정이 완료되면 AccessManager는 ObjectStore의 API를 이용할 수 있다.

3.4 AccessManager의 환경설정(Configuration)

기존의 Jini 서비스에서 추가된 사항 중 개발자에게 응용프로그램을 테스팅하고 배치할 때 다양한 편의를 제공하는 부분이 환경설정 파일이다. AccessManager에 대한 환경설정은 다음과 같다. 먼저, 자바의 JAAS 로긴을 통해 AccessManager.login 파일을 설정하고, 공개키 인증서를 갖고있는 truststore 파일의 위치정보를 지정한다. truststore 파일 내의 인증서를 검색하여 각 사용자에 대한 공개키 인증을 수행한다. 이는 AccessManager 서비스에 접근할 수 있는 사용자를 "client", "reggie", "ObjectStore"로 한정시킨다.

```
Server.AccessManager {
    /* JAAS login */
    loginContext = new LoginContext("AccessManager.jaas.login");

    /* Keystore for getting principals */
    private static caTruststore = KeyStores.getKeyStore("${installDir}
        ${/}example${/}common${/}jsse${/}truststore${/}jini-ca.truststore",
        null);

    /* Retrieve from the truststore above, the public key certificate */
    private static clientKey = KeyStores.getX500Principal("client", caTruststore);
    private static reggieKey = KeyStores.getX500Principal("reggie",
                                caTruststore);
    private static AccessManagerKey = KeyStores.getX500Principal
                                ("AccessManager", caTruststore);
    private static outriggerKey = KeyStores.getX500Principal("ObjectStore",
                                caTruststore);
```

(그림 10) AccessManager.cofig 파일

3.4.1 서비스 추출기(Exporter)

추출기(exporter)를 구현하는데 사용하는 단말(endpoint)을 생성한다. AccessManager는 JERI기반의 SSL 프로토콜을 사용하도록 설계하였으므로, 서버 측에서 SSL 프로토콜을 사용할 수 있도록 JERI/JSSE 서버 단말을 설정한다.

```
/* Exporter ---------------------------------- */
  private serviceEndpoint = SslServerEndpoint.getInstance(0);
  private serviceConstraints = new BasicMethodConstraints
                    (new InvocationConstraints (new InvocationConstraint[]
                    { Integrity.YES }, null));
  private serviceILFactory = new ProxyTrustILFactory(serviceConstraints,
                                                    OutriggerPermission.class);
  serverExporter = new BasicJeriExporter(serviceEndpoint, serviceILFactory);
```

(그림 11) 서비스 추출기

3.4.2 프락시 준비기(Proxy Preparer)

AccessManager는 ObjectStore 서비스를 이용하기 위하여 서비스프락시를 다운로드 받는다. (그림 12)은 다운로드 받은 프락시에 대하여 프락시준비기 수행을 보여주고 있다. 신뢰성이 확인된 프락시는 무결성, 클라이언트 인증, 그리고 서버인증 제한을 설정한다.

```
/* Preparer for ObjectStore proxy */
   static preparer = new BasicProxyPreparer( true,
     new BasicMethodConstraints(
          new InvocationConstraints(
            new InvocationConstraint[] { Integrity.YES,
                 ClientAuthentication.YES, ServerAuthentication.YES,
                 new ServerMinPrincipal(outriggerPublicKeyCert) }, null)),
      new Permission[] {
          /* Authenticate as client when connecting to server */
          new AuthenticationPermission(clientPublicKeyCert,
                                  outriggerPublicKeyCert, "connect") });

   private groups = new String[] { "JSSE_Group0", "JSSE_Group1" };
   serviceDiscovery = new ServiceDiscoveryManager(
       new LookupDiscovery(groups, this), null, this);
}
```

(그림 12) 프락시 준비기

3.4.3 AccessManager의 보안관련 파일 설정

Jini 기반의 프로그래밍에서 클라이언트와 서버 사이의 인증을 위하여 JDK 내의 키툴(keytool)을 이용하여 키를 생성한다. 또한 키툴을 이용하여 키와 인증서, 비밀번호를 저장하는 키저장소(keystore)를 생성한다[10]. 이러한 정보는 Jini2.0 보안모델로써 *AccessManager.login* 파일에 그 위치정보를 기술해야 한다.

```
AccessManager.jaas.login {
     com.sun.security.auth.module.KeyStoreLoginModule required
    keyStoreAlias = "AccessManager"
    keyStoreURL = "file:${installDir}${/}example${/}common${/}jsse${/}keystore
                    ${/}accessmanager.keystore"
    keyStorePasswordURL = "file:${installDir}${/}example${/}common${/}jsse
                            ${/}keystore${/}accessmanager.password";
};
```

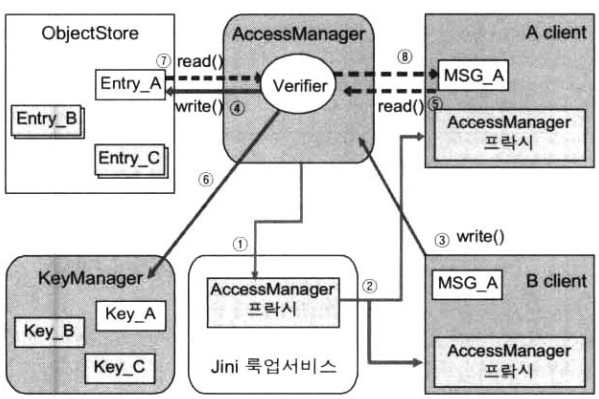
(그림 13) AccessManager.login 파일

JavaSpace를 이용하는 사용자에 따른 권한을 설정해주는 AccessManager.policy 파일을 작성해야한다. policy 파일 내에는 사용자에 대하여 메소드별 권한을 설정할 수 있다.

4. SecureJS 시스템의 활용예제

분산 컴퓨팅 환경에서 JavaSpace 서비스는 객체를 공유하고 교환하는 메커니즘으로 다양한 분야에서 사용되고 있다. 기존의 JavaSpace는 객체를 저장하거나 저장된 객체에 접근하고자 할 경우 보안성이 매우 부족하여 보안을 요구하는 응용프로그램에서는 사용하기에 부적합하였다. 이러한 문제점을 해결하기 위해 Jini2.0 보안모델을 이용하여 JavaSpace 서비스와 사용자 사이의 상호인증을 수행하였으며 또한 JavaSpace 서비스를 제공하는 ObjectStore를 유일하게 접근할 수 있는 AccessManager를 설계하였다. AccessManager는 ObjectStore에 대한 접근권한을 위임받아 클라이언트에게 안전한 JavaSpace 서비스를 제공한다.

기존의 JavaSpace 서비스를 메시징 시스템에 활용한다면 다양한 문제점을 가질 수 있다. 예를 들어, 특정 수신자를 지정할 수 없으므로 누구든지 저장된 메시지 객체에 대하여 접근할 수 있으며, 악의의 수신자는 그 메시지 객체를 삭제할 수도 있다. 그래서 JavaSpace는 보안을 요구하는 메시징 시스템에는 적용하기가 힘들었었다. (그림 14)는 SecureJS 시스템을 이용하여 안전하게 메시지를 전송하고 수신하는 과정을 보여주고 있다.



(그림 14) SecureJS를 이용한 메시징프로그램

① AccessManager는 Jini 서비스로 동작하기 위하여 자신의 서비스프락시를 룩업서비스에 등록시킨다.

② 클라이언트들은 ObjectStore에 접근하기 위하여 AccessManager의 서비스프락시를 다운로드 받아야한다.

③ 그림에서와 같이 송신자 B가 수신자 A에게 Entry_A라는 메시지 객체를 ObjectStore를 통하여 보낸다. 송신자 B는 AccessManager의 write() 메소드를 사용한다.

④ AccessManager는 메시지를 받아야할 수신자 A의 id를 암호화시킨 값과 메시지 객체를 엔트리로 구현하여 ObjectStore에 저장한다.

⑤ 수신자 A가 메시지를 읽기 위해 AccessManager의 read() 메소드를 호출한다.

⑥ AccessManager의 검증자(Verifier)는 수신자 A가 메시지 객체인 MSG_A에 접근할 수 있는 올바른 수신자인지 여부를 검사하기 위하여 키 관리자에서 수신자 A의 공개키를 구한다.

⑦ 수신자 A가 올바른 수신자임이 검증되면 read() 메소드를 이용하여 ObjectStore 내의 Entry_A를 읽어온다.

⑧ 읽어온 엔트리 객체에서 MSG_A를 추출하여 수신자에게 전달한다.

다음 소스코드는 메시지를 보내는 송신자 프로그램에 대한 예제이다(Client_A.java).

```
public class Client_A {
    private AccessManager secure_JS ;
    public static void main(String[] args) throws Exception {
        final Configuration config = ConfigurationProvider.getInstance(args) ;
        LoginContext loginContext = (LoginContext) config.getEntry
                ("client.Client_A", "loginContext", LoginContext.class, null) ;
                                :
                                :
    /* Look up the AccessManager Demon */
            ServiceItem serviceItem = serviceDiscovery.lookup(
                new ServiceTemplate(null, new Class[] { AccessManager.class
                }, null), null, Long.MAX_VALUE) ;
            secure_JS = (AccessManager) serviceItem.service ;

    /* Prepare the AccessManager proxy */
            ProxyPreparer preparer = (ProxyPreparer) config.getEntry(
                                    "client.Client_A", "preparer",
                                     ProxyPreparer.class, new
                                     BasicProxyPreparer()) ;
            secure_JS = (AccessManager) preparer.prepareProxy(secure_JS) ;

    /* Use the AccessManager Services */
①  ------  ldap_pub_key = ldapCtx.search(receiver_id, PK_FILTER, con-
                        straints) ;
②  ------  Message msg_A = new Message() ;
            msg_A.content = ("Hello, Jini!!") ;
③  ------  secure_JS.write(receiver_id, msg_A, null, Lease.FOREVER) ;
    }
}
```

(그림 15) 예제 프로그램 : Client_A.java

① 키 관리자로부터 수신자의 공개키를 구한다.

② Client_B에게 전송할 메시지 객체를 생성한다.

③ SecureJS 시스템의 write() 메소드를 호출하여 메시지 객체를 저장한다.

(그림 16)은 메시지를 받는 수신자 프로그램이다. 위의 코드에서 룩업서비스와 AccessManager의 프락시를 가져오는 코드부분은 동일하여 생략하였다(Client_B.java).

① 자신의 개인키로 id를 암호화시킨다.
② 수신할 메시지 객체를 JavaSpace로부터 찾기 위하여 템플릿을 작성한다.
③ SecureJS 시스템의 read() 메소드를 호출하여 자신이 올바른 수신자임을 증명하고, 읽어야할 객체를 템플릿을 통하여 얻는다.

```
public class Client_B {
    private AccessManager secure_JS ;
    public static void main(String[ ] args) throws Exception {
        final Configuration config = ConfigurationProvider.getInstance(args) ;
        LoginContext loginContext = (LoginContext) config.getEntry
                ("client.Client_B", "loginContext", LoginContext.class, null) ;
                                :
                                :
    /* Use the AccessManager Services */
① ------ private_encrypt_id = encrypt_my_id(private_key, id) ;
② ------ Message template = new Message() ;
③ ------ Message result = (Message)secure_JS.read(private_encrypt_id, id,
                                template, null, Long.MAX_VALUE) ;
        System.out.println(result.content) ;
    }
}
```

(그림 16) 예제 프로그램 : Client_B.java

5. 결 론

Jini 시스템은 개발자에게 분산시스템을 쉽게 개발할 수 있는 하부구조를 제공한다. 그러나, 보안성이 취약한 기존의 Jini 서비스는 보안을 요구하는 분산응용프로그램 개발에 있어서 보안은 개발자의 영역으로 남겨두었었다. 이러한 문제를 개선하기 위하여 최근 썬 마이크로시스템즈(Sun Microsystems)에서는 자바코드의 이동성에 따른 보안요소를 충족시키는 Jini2.0을 제시하였다. 그 결과 Jini2.0을 이용하여 보안을 요구하는 분산응용프로그램을 보다 편리하게 작성할 수 있게 되었다.

Jini 서비스 중에 하나인 JavaSpace는 표준 인터페이스를 통하여 객체를 저장하고, 저장된 객체를 그 클래스의 템플릿을 이용하여 읽거나 가져오는 패러다임을 갖고 있다. 이러한 특징은 분산시스템을 개발할 때, 분산객체의 영속성과 데이터 교환 기능을 지원하는 시스템을 구현하는데 많이 활용되고 있다. 그러나, 기존의 JavaSpace는 보안기능이 취약하여 누구나 객체를 저장하고 저장된 객체를 제약 없이 누구든지 접근할 수 있었다. 그러므로 보안이 요구되는 정보를 교환하거나 보관할 경우 JavaSpace는 객체 저장소로서의 서비스를 제공하기 힘들다.

본 논문에서는 새로운 Jini2.0 보안모델과 프로그래밍 모델을 적용하여 JavaSpace 서비스를 안전하게 제공할 수 있는 SecureJS 시스템을 개발하였다. SecureJS 시스템은 AccessManager와 ObjectStore 그리고 KeyManager로 구성되어 있다. ObjectStore는 JavaSpace 서비스를 제공하는 객체저장소 이며, LDAP를 이용한 KeyManager는 AccessManager에서 올바른 수신자인지 여부를 검사할 때 사용하는 공개키를 관리한다. AccessManager는 인증된 클라이언트에게 안전한 JavaSpace 서비스를 제공하기 위하여 ObjectStore에 대한 모든 접근권한을 위임받아 서비스한다. 또한 클라이언트는 AccessManger와 인증이 되었다 할지라도 JavaSpace 서비스를 요구하면 해당 엔트리에 대하여 검증된 수신자임을 검사한 다음 서비스를 제공하도록 구현하였다. 그 결과 개발된 Jini2.0 기반의 SecureJS 시스템은 분산 컴퓨팅 환경에서 객체를 안전하게 공유하고 저장할 수 있는 매우 유용한 기술을 제공하고 있다. 그래서 본 시스템의 활용 및 응용영역은 매우 다양할 것으로 여겨진다.

추후 연구는 이전 연구에서 개발된 이동 에이전트시스템인 JMoblet 시스템을 보안성이 강화된 Jini2.0 시스템 환경으로 개선할 것이다[15, 16]. 주된 기능은 SecureJS 시스템을 적용하여 보안을 요구하는 이동 에이전트간 통신 시 안전한 통신기능을 지원하고[17], 또한 이동 에이전트의 안전한 저장소로써의 기능을 제공하는 SecureJS 기반의 이동 에이전트 시스템을 개발할 것이다.

참 고 문 헌

[1] Sun Microsystems, "Jini™ Architecture Specification," Published Specification, http://java.sun.com/products/jini/2.0/doc/specs/html/jini-spec.html, 2003.

[2] Sun Microsystems Inc, "Jini Technology Core Platform Specification," Communications of the ACM, Vol.39, No. 4, pp.75-83, 1996.

[3] Sun Microsystems, "Jini™ Technology Starter Kit Overview v2.0," Published Specification, http://java.sun.com/developer/products/jini/arch2_0.html, 2003.

[4] Dan Creswell, "Getting started with JINI™ 2.0," Dan Creswell, http://www.dancres.org/cottage/starting_jini.html, 2003.

[5] Jan Newmarch, "Jan Newmarch's Guide to Jini Technologies," Manning Publications Co., 2003.

[6] Sun Microsystems, "JavaSpaces™ Service Specification," Published Specification, http://wwws.sun.com/software/jini/specs/jini1.2html/js-title.html, 2002.

[7] Frank Sommers, "Jini Starter Kit 2.0 tightens Jini's security framework," Los Alamitos, CA., IEEE Computer Society Press, 2003.

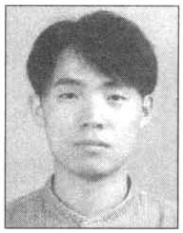
[8] G. P. Picco, A. L. Mruphy and G-C. Roman, "Lime : Linda Meets Mobility," Ind.Garlan, editor, Proc. of the 21stInt. Conf. on Software Engineering, pp.368-377, 1999.

[9] Sun Microsystems, "JavaSpaces™ v2.0 API Documentation," Published Specification, http://java.sun.com/prod-

ucts/jini/2.0/doc/api/net/jini/space/JavaSpace.html, 2003.
[10] Sun Microsystems, "Security enhancements for the Java2 SDK," http://java.sun.com/j2se/1.4.2/docs/guide/security/index.html, 2003.
[11] Sun Microsystems, "Secure Computing with Java : Now and the Future," http://java.sun.com/security/javaone97-whitepaper.html. 1997.
[12] W. Yeong, T. Howes and S. Kille, "Lightweight Directory Access Protocol," RFC 1777, March, 1995.
[13] Rob Weltman, Tony Dahbura, "LDAP Programming with Java," Addison-Wesley, 2000.
[14] 문남두, 안건태, 박양수, 이명준, "그룹통신을 이용한 견고한 LDAP 서버", 정보처리학회논문지C, 제10-C권 제2호, 2003.
[15] 김진홍, 구형서, 유양우, 이명준, "JMoblet : Jini 기반의 이동에이전트시스템", 정보처리학회논문지B, 제8-B권 제6호, pp.292-312, 2001.
[16] 유양우, 문남두, 이명준, "SecureJMoblet : 안전한 Jini 기반의 이동 에이전트 시스템", 한국정보과학회 춘계학술발표회, pp.562-564, 2004.
[17] 유양우, 이명준, "분산응용프로그램을 위한 안전한 JavaSpace", 한국정보과학회 춘계학술발표회, pp.352-354, 2003.

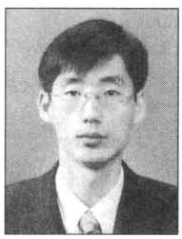
유 양 우

e-mail : soft@mail.uc.ac.kr
1995년 경일대학교 전자계산학과(공학사)
1997년 울산대학교 대학원 전자계산학과 (공학석사)
2000년 울산대학교 대학원 전자계산학과 박사과정 수료
2000년~현재 울산과학대학 컴퓨터정보학부 전임강사
관심분야 : 분산객체 시스템, 이동에이전트 시스템, 웹기반 정보시스템 등



문 남 두

e-mail : dooya@ulsan.ac.kr
1997년 울산대학교 전자계산학과(공학사)
1999년 울산대학교 컴퓨터정보통신 공학부 (공학석사)
2003년 울산대학교 컴퓨터정보통신 공학부 (공학박사)
관심분야 : 그룹통신 시스템, 분산객체, CSCW, 웹 프로그래밍 등



정 혜 영

e-mail : hyjung@ulsan.ac.kr
1994년 울산대학교 전자계산학과(공학사)
1998년 울산대학교 대학원 전자계학학과 (공학석사)
1993년~1995년 (주)현대중공업 조선사업부 근무
1997년~2003년 (주)LG CNS 근무(휴직중)
2000년~현재 울산대학교 대학원 컴퓨터정보통신공학부 박사과정
2003년~현재 울산대학교 컴퓨터정보통신공학부 객원교수
관심분야 : WebDAV, 협업지원시스템, 분산 컴퓨팅 등

이 명 준

e-mail : mjlee@ulsan.ac.kr
1980년 서울대학교 수학과(학사)
1982년 한국과학기술원 전산학과(석사)
1991년 한국과학기술원 전산학과(박사)
1993년~1994년 미국 버지니아대학 교환교수
1982년~현재 울산대학교 컴퓨터정보통신공학부(교수)
관심분야 : 웹기반 정보시스템, 프로그래밍언어, 분산 프로그래밍, 생물정보학