푸시기반 CORBA 트레이더 서비스 구현
(A Push-based Implementation of CORBA Trader Service)

유 재 정 † 윤 범 렬 †† 김 수 동 †††
(Jae Jeong You) (Bum Ryoel Yoon) (Soo Dong Kim)

요 약 트레이더 서비스는 서비스를 이용하고자 하는 객체가 서비스를 제공하는 객체에 관한 사전 지식이 없더라도 서비스의 특성에 따라 가장 적절한 서비스를 찾아 이용할 수 있게 한다. 이를 위해 트레이더 서비스는 임포터가 요청한 서비스를 임포터, 트레이더, 링크의 정책에 따라 매칭되는 서비스를 지역 트레이더나 타 트레이더와의 연합을 통해 서비스 오퍼를 가공하여 전달한다. 이러한 기존의 트레이더 방식은 임포터가 요청한 서비스 오퍼의 양과 트레이더간 연합에 따라 요청에 대한 결과가 늦게 전달됨으로써 사용자의 대기시간이 길어지고 트레이더의 성능을 저하시킨다. 정확한 서비스 오퍼와 빠른 서비스 제공을 위해 트레이더 서비스의 성능 문제는 트레이더 서비스 구현 시 중요한 기준이다.

본 논문에서는 트레이더 서비스의 성능 저하 문제를 해결하고 최상의 서비스를 제공하기 위해 OMG에서 정의한 임포터와 트레이더의 정책에 새로운 정책을 추가하고, 빠른 서비스 제공을 위한 트레이더 서비스의 향상된 PUTS(PUsh Trader Service) 모델을 제시한다. 또한 모델의 주요 모듈 설계 및 구현을 제시하고, 제안한 모델을 이용하여 트레이더 서비스 시스템을 구현한다. 구현 시스템의 성능 평가를 위해 일반적인 트레이더 서비스 시스템의 유형별로 분석, 평가한다.

Abstract CORBA Trader Service is to locate appropriate objects that provide the desired functionality at runtime. To provide this service, the Trader Service federates a local trader and remote traders by considering the Traders or Link policies, and it returns the service offers that are requested by the importer. This traditional way of trading reveals a performance problem due to the low response time. The response time largely depends on the amounts of service offers of the Importer' s request and the frequency of federations with the other trader. The performance is a key factor for evaluating the Trader Service performance.

In order to overcome the low response time and to provide the high-quality services, we propose new policies of Import and Trader, and present a PUTS(Push Trader Service) model which implements this new advanced trader service. We present the design and implementation of the PUTS' s major modules, also make a comparison between PUTS system and traditional trader system in terms of performance and functionality through case studies.

1. 연구 배경

CORBA(Common Object Request Broker Architecture)[1]의 등장은 클라이언트/서버의 분산 환경에서의 상호연동성과 분산 어플리케이션의 확장을 가져왔으나, CORBA 기반의 분산 객체 어플리케이션의 개발은 객체의 분산 및 이들 객체간의 상호 연동을 고려해야 하므로 일반 어플리케이션보다 개발하기가 어렵다. 따라서 OMG(Object Management Group)에서는 CORBAservices[2] 명세에서 분산 어플리케이션의 확장과 개발을 위한 여러 서비스를 정의하고 있다. 특히 CORBAservices 명세의 일부로서 원하는 객체를 쉽게 발견할 수 있도록 네이밍 서비스(Naming Service)와 트레이더 서비스(Trader Service)를 정의하고 있다. 네이밍 서비스는 객체의 이름으로 서비스 객체를 검색하

† 비 회 원 : 숭실대학교 전자계산학과
mryoujj@chollian.net
†† 학생회원 : 숭실대학교 전자계산학과
bryoon@object.soongsil.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학부
dskim@computing.songsil.ac.kr
논문접수 : 1999년 3월 11일
심사완료 : 1999년 10월 19일

는 방식인 반면 트레이더 서비스는 객체의 특성(Property)을 가지고 객체를 검색하는 방식이다[2]. 트레이더 서비스에서 객체는 이름을 가지고 있지 않으며 서비스의 유형(Service Type)에 따라 관리된다. 따라서, 서비스를 제공하기 위해서 서버는 트레이더에 서비스의 유형, 특성을 엑스포트(Export)하고, 서비스를 받고자 하는 클라이언트는 트레이더에 원하는 서비스의 정보를 임포트(Import)한다. 이 때, 서비스를 엑스포트하는 객체를 엑스포터(Exporter)라 하고 서비스를 요구하는 객체를 임포터(Importer)라 하며 전송되는 서비스 정보를 서비스 오퍼(Service Offer)라 한다[2].

트레이더 서비스에서 클라이언트는 원하는 서비스를 찾기 위해 여러 가지 정보를 입력하여 트레이더에 요청하게 된다. 즉, 서비스 유형, 제약 사항(Constraint), 우선 조건(Preference), 특성의 종류(Desired Property) 등의 정보를 입력한다[3,4]. 이 외에도 검색할 서비스 오퍼(Service Offer)의 수나 전송될 서비스 오퍼의 수, 링크의 행위 규칙(Link Follow Rule) 등 임포터의 정책을 입력한다. 이렇게 클라이언트로 하여금 많은 정보를 요구하는 것은 많은 서비스 오퍼가 존재하는 분산 환경에서 가장 정확한 서비스를 검색하고 최소한의 필요한 정보만을 전송받기 위한 부가 장치라 할 수 있다. 하지만 클라이언트가 서비스의 유형이나 특성을 정확하게 알지 못한 상태에서 트레이더에 서비스를 요청한 경우, 매칭되는 서비스 오퍼를 모두 실시간으로 클라이언트에 전송하면 사용자의 대기시간은 매우 길어지게 된다. 특히, 원하는 서비스 유형을 지역 트레이더가 관리하지 않고 다른 도메인의 트레이더가 관리하고 있다면 이들 트레이더간의 연합(Federation)에 따른 지연 시간까지 합하면 사용자의 대기시간은 더욱 늘어나게 된다[5,6]. 이러한 대기시간의 문제는 트레이더 서비스를 이용하는 사용자로 하여금 트레이더 서비스의 성능을 의심하게 할 수 있다.

기존의 연구는 트레이더 서비스의 성능 향상을 위해 연합을 효율적으로 수행하기 위한 방법에 한정하였다. 즉, 서비스 오퍼 공간을 효율적으로 분할하여 트레이더간 연합을 최소화하는 방법이 주로 연구되어 왔다[7]. 하지만, 본 논문에서는 이러한 대기시간의 문제와 연합에 따른 서비스 지연의 문제를 해결하기 위해 OMG의 정책과 트레이더 서비스의 아키텍쳐 측면을 고려하여 향상 방법을 연구한다. 즉, 기존의 트레이더 서비스의 정책에 새로운 정책을 추가하여 제안하고 클라이언트의 유휴시간에 요청한 서비스의 정보를 미리 클라이언트로 푸시(Push)함으로써 서비스를 빠르게 검색할 수 있는 모델(PUTS)을 제안한다. 또한 모델의 주요 구성 모듈에 대한 설계 및 구현을 제시하고, 제안된 모델을 적용한 트레이더 시스템의 개발 사례와 성능을 비교한다.

본 논문은 다음과 같이 구성된다. 2장에서는 향상된 트레이더 시스템을 개발하기 위한 관련 기술을 설명한다. 3장에서는 트레이더 서비스의 기능 향상을 위한 기법을 정책적 측면과 아키텍쳐 측면에서 기술하고 주요 구성 모듈의 설계 및 구현을 제시한다. 4장에서는 제안된 모델을 적용한 트레이더 시스템의 개발 사례와 시스템의 성능을 평가하고, 마지막으로 5장에서는 결론 및 향후 연구 과제에 관하여 기술한다.

2. 관련 기술

2.1 트레이더 연합

트레이더는 분산환경에서 서비스를 제공하는 객체와 이를 이용하려는 객체간에 위치하여 엑스포트된 서비스의 정보(서비스 오퍼)를 관리한다. 트레이더는 하나나 그 이상의 서비스 유형별로 데이타베이스에 서비스 오퍼를 관리하고, 서비스 오퍼의 수가 많아질수록 트레이더의 부하는 매우 커지게 된다[8,9]. 따라서, 하나의 트레이더는 모든 서비스 유형을 관리하기 보다는 다른 서비스 유형을 관리하는 트레이더와의 연합을 통해 트레이더의 부하를 줄이게 된다. 트레이더간 연합은 동적으로 다양하게 변하는 분산환경에서 클라이언트에게 최상의 서비스를 제공하고, 서비스 오퍼의 분산을 통한 서버의 부하 감소 측면에서 트레이더의 기능 중 중요한 부분을 차지한다.

트레이더의 형태에는 질의 트레이더(Query Trader), 단순 트레이더(Simple Trader), 스탠드올런 트레이더(Stand-alone Trader)와 같은 간단한 형태가 있다[2]. 질의 트레이더는 Lookup 인터페이스만을 지원하는 트레이더이고, 단순 트레이더는 Lookup, Register인터페이스를 지원하는 트레이더이며, 스탠드올런 트레이더는 Lookup, Register, Admin인터페이스를 지원하는 트레이더이다. 이러한 간단한 형태의 트레이더는 한 도메인 내에서 서비스 객체를 찾아주는 지역 트레이더 역할을 담당한다. 반면 다른 도메인의 트레이더와의 연합을 위해서는 위 세가지 인터페이스 외에 Link 인터페이스가 추가된 연결형 트레이더(Linked Trader)가 요구된다. 이 외에도 좀 더 복잡한 형태의 프락시 트레이더(Proxy Trader), 완전-서비스 트레이더(Full-service Trader)가 있다[2].

타 트레이더와의 연합은 연결형 트레이더나 프락시 트레이더, 완전-서비스 트레이더의 형태를 띤 트레이더

간에 서비스 오퍼가 교환되는 것을 의미한다.

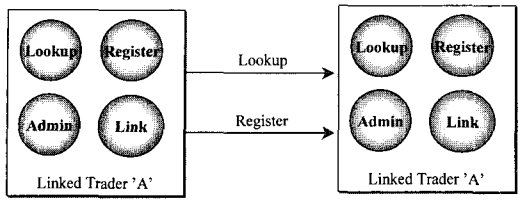

그림 1 연결형 트레이더의 연합 형태

그림 1은 연결형 트레이더에서 지원하는 인터페이스와 트레이더간 연합의 형태를 나타내고 있다. 트레이더A는 클라이언트가 요청한 서비스 질의문을 트레이더B의 Lookup인터페이스에 요청한다. 이때 어떤 트레이더와 연합을 수행할지는 Link인터페이스에서 관리하고 있는 링크의 종류에 의해 결정된다.

트레이더 연합의 형태는 위의 그림 1처럼 단순한 형태의 협력 형태(Light Weight Cooperation) 이외에도 협의 형태(Simple Negotiation), 연합 형태(Federation) 형태로 구분된다[8].

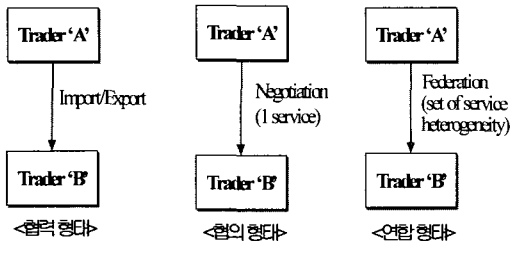

그림 2 트레이더 연합의 형태

협력 형태는 트레이더A가 임포트나 엑스포트를 통해 트레이더B에 질의를 수행하거나 자신을 등록하는 과정이다. 반면 협의 형태에서 트레이더A는 클라이언트가 요구하는 서비스 오퍼를 트레이더B에도 요청하고 전달된 서비스 오퍼에 대한 정보를 통합하여 클라이언트에 전달해 주는 형태이다. 이 경우 서비스 사용에 대한 빈도는 협동 형태보다 매우 많게 된다. 마지막으로 연합 형태는 하나의 서비스 유형 뿐 만 아니라 서로 다른 서비스 유형간 서비스 이용 빈도가 매우 높은 복잡한 형태의 연합이 수행되는 유형이다.

협의와 연합의 형태에서는 트레이더간 정보 교환 빈도가 매우 높고 연합이 수시로 수행되므로 서비스 오퍼를 클라이언트에 전송하기까지의 시간은 크게 증가한다. 이처럼 연합의 수(Hop Count)나 전송할 서비스 오퍼의 수가 많을수록 클라이언트의 대기시간은 길어진다. 따라서, 사용자에게 최상의 서비스를 제공하기 위해서는 연합에 따른 대기시간 문제점을 해결해야만 한다.

2.2 스마트 에이전트와 스텁

스마트 에이전트(Smart Agent)[10,11]는 클라이언트 어플리케이션과 구현 객체간에 동적인 분산 디렉토리 서비스를 제공함으로써, 서로 다른 도메인 간(Inter-Domain)의 트레이더 연합을 가능하게 한다. 각 도메인의 에이전트는 타 도메인의 IP를 관리하면서, 설정된 IP의 에이전트와 상호 연동을 수행한다. 트레이더의 연합을 위해 지역 트레이더는 다른 도메인에 존재하는 트레이더의 Lookup 객체와 바인딩(Binding)하고 Lookup객체의 주소(Reference)를 얻는다. 이때 타 도메인의 Lookup 객체의 이름을 찾아 바인딩 역할을 수행하는 것이 스마트 에이전트이다.

스텁(Stub)은 클라이언트 상에 위치하여 원격 객체를 접근하기 위한 접근 통로 역할을 제공하는 프락시 객체이다[11,12]. 스텁은 클라이언트가 요구한 원격 객체에 GIOP(General Inter-ORB Protocol)[1] 형태로 마샬링(Marshalling)하여 메시지를 전송하고, 원격 객체에서 전달되어진 응답 메시지를 언마샬링(Unmarshalling)하여 클라이언트 객체에 넘겨주는 역할을 한다[12]. 따라서, 클라이언트가 요청한 모든 메시지와 응답 결과는 스텁을 통하게 되므로 스텁을 적절한 형태로 수정하면 클라이언트 객체와의 다양한 연동을 가능하게 할 수 있다.

2.3 연동 가능 객체 주소 (IOR: Interoperable Object Reference)

엑스포터가 트레이더에 서비스를 등록할 때는 엑스포트하는 객체의 주소, 서비스 유형, 하나 또는 그 이상의 특성을 입력해야 한다. 엑스포터는 트레이더의 Register 인터페이스의 export() 메소드를 통해 자신의 서비스를 등록한다. 등록이 성공하면 트레이더로부터 지역 트레이더에서 유일한 오퍼 인식자(Offer Identifier)를 받게 된다. 이후 엑스포터는 전달받은 오퍼 인식자를 통해 자신

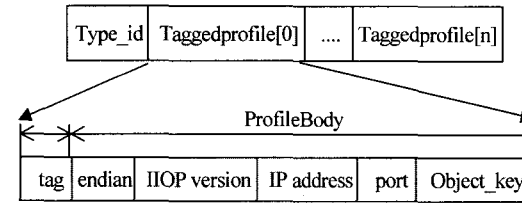

그림 3 연동가능 객체 주소의 구조

의 서비스를 삭제하거나 수정할 수 있게 되며, 임포터는 트레이더에 등록된 특성과 서비스의 종류에 따라 엑스포터의 주소를 얻게 된다. CORBA에서 모든 객체의 주소는 그림 3과 같이 구성된다.

3. 성능 향상 기법

트레이더의 연합에 따른 지연 시간 문제를 해결하고 최상의 서비스를 제공하기 위한 해결방안으로 푸시형 트레이더 서비스(PUTS) 모델을 제안한다. 즉, 클라이언트는 원하는 서비스 오퍼를 실시간으로 전송받거나 또는 클라이언트의 유휴시간에 트레이더로부터 서비스 오퍼를 클라이언트의 캐쉬 데이타베이스에 전송받아 지역 데이타베이스에서 검색을 수행하게 하는 모델이다. 본 모델은 푸시 기능을 제공하지 않는 다른 트레이더와의 연동을 위해 기존 트레이더의 Lookup, Register, Link, Admin 등의 기본 인터페이스를 지원하고, 부가적으로 클라이언트가 요구하는 서비스 오퍼를 미리 전송받을 수 있는 새로운 인터페이스와 정책을 추가한다.

표 1 트레이더 서비스의 정책

유형	정 책 명	구 분
검색 범위 정책	def/max_search_card	트레이더
	def/max_match_card	트레이더
	def/max_return_card	트레이더
	def/max_hop_count	트레이더
	def/max_follow_policy max_link_follow_policy	트레이더
	search_card	임포터
	match_card	임포터
	return_card	임포터
	hop_count	임포터
	link_follow_rule	임포터
	starting_trader	임포터
	request_id	임포터
	exact_type_match	임포터
	def_pass_on_follow_rule	링크
	limiting_follow_rule	링크
기능 결정 정책	support_modifiable_properties	트레이더
	support_dynamic_properties	트레이더
	support_proxy_offers	트레이더
	use_modifiable_properties	임포터
	use_dynamic_properties	임포터
	use_proxy_offers	임포터

3.1 성능 향상 정책

OMG의 트레이더 서비스 명세에서 정의한 정책의 유형에는 검색의 범위를 지정하는 정책(Scoping Policies)과 오퍼레이션에 적용될 기능을 결정하는 정책(Capability Supported Policies)이 있다[2]. 검색의 범위를 지정하는 정책은 다시 임포터 정책, 트레이더 정책, 링크 정책으로 구분된다.

표 1은 트레이더 서비스 명세에서 정의한 정책을 유형별로 구분한 것이다. 클라이언트는 임포터와 관련된 각 정책을 Policy 객체로 정의한 후 이러한 Policy 객체의 배열을 Lookup 인터페이스의 query() 메소드의 파라미터로 전송한다. 트레이더는 전송된 임포터의 정책과 트레이더 정책 범위를 비교하고 정책명의 오류를 체크함으로써 검색의 범위나 기능을 결정한다.

위의 정책 이외에 클라이언트가 요청한 서비스 정보가 푸시해야할 정보인지 아니면 바로 검색을 실행하여 전송해 줄 정보인지를 판단하기 위해 PUTS모델에서는 트레이더와 임포터에 각각 새로운 정책을 요구한다.

표 2 푸시 기능을 위한 정책 정의

유 형	정 책 명	구 분	IDL Type
검색 범위 정책	user_ID	임포터	string
기능 결정 정책	support_use_push	트레이더	boolean
	use_push	임포터	boolean

표 2는 푸시 기능을 수행하기 위해 요구되는 새로운 정책을 정의한 것이다. 임포터의 'use_push' 정책은 클라이언트가 요청하는 서비스에 대해 푸시 기능의 수행 여부를 결정한다. 즉, 'use_push'를 'true'로 설정하여 전송하면 요청한 서비스 오퍼를 즉시 전송하지 않고 클라이언트의 유휴 시간에 트레이더가 클라이언트의 캐쉬 데이타베이스에 푸시해 준다. 이 때 어떤 클라이언트가 푸시를 요청하였는지를 판단하고 이후 해당 클라이언트에 서비스 오퍼를 푸시하기 위해 문자열 타입의 'user_ID'정책이 필요하다. 결국 클라이언트는 푸시 기능의 실행을 위해 'user_ID'와 'use_push'를 Policy 객체로 만들어 전송한다.

트레이더의 'support_use_push' 정책은 트레이더의 푸시 기능 수행 여부를 결정한다. 만약 이 값이 'true'로 설정되어 있으면 트레이더가 클라이언트에 푸시 기능을 지원하고, 'false'로 설정되었거나 이러한 정책이 존재하지 않으면 일반 트레이더처럼 클라이언트의 요청에 대

한 결과를 바로 전송하게 된다.

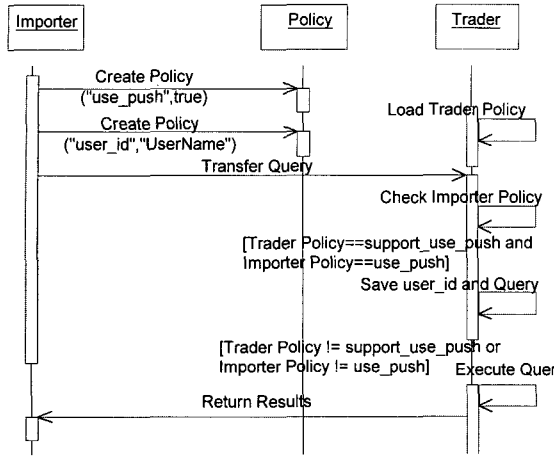

그림 4 푸시 정책 수행 순차도

그림 4는 앞에서 정의한 임포터와 트레이더의 푸시 정책에 따른 순차도(Sequence Diagram)[13]를 나타낸다. 트레이더는 자신의 정책과 임포터의 정책에 따라 푸시 기능을 수행할 경우, 트레이더의 데이타베이스에 임포터의 질의에 포함된 정보를 저장하고 이후 검색 에이전트(Search Agent)[14,15]가 내부적으로 구동되면서 검색을 실행하고 그 결과를 클라이언트에 푸시한다.

3.2 아키텍쳐

새로운 트레이더 서비스 정책은 클라이언트가 질의를 요청할 때 설정한 정책 및 질의 정보를 일단 서버의 데이타베이스에 저장한 후 검색 에이전트에 의해 검색을 실행하고 검색 결과를 클라이언트의 캐쉬 데이타베이스에 푸시한다.

그림 5는 이러한 기능을 수행하기 위한 트레이더 서비스의 구성도를 나타낸다.

클라이언트 상에는 서비스를 요청한 임포터 객체 또는 GUI와 스텁, 푸시 관리자(Push Manager), 클라이언트 어댑터(Client Adapter), 전송 에이전트(Transfer Agent) 모듈이 존재하고, 트레이더 상에는 연결형 트레이더 기능을 제공하기 위한 기본적인 인터페이스와 트레이더 어댑터(Trader Adapter), 검색 에이전트(Search Agent)로 구성된다.

3.2.1 GUI 관리 모듈

클라이언트 측의 임포터가 엑스포터가 제공하는 서비스를 받기 위해서는 먼저 트레이더가 관리하는 서비스 오퍼를 찾아야 한다. 이를 위해 원하는 서비스 유형을 식별하고, 서비스 오퍼에서 가장 적절한 서비스 객체를 찾을 수 있도록 제약 사항, 우선 조건, 특성의 종류 등의 정보를 용이하게 입력할 수 있는 기능이 요구된다. 이 외에도 검색할 서비스 오퍼의 수나 전송될 서비스 오퍼의 수, 링크의 행위 규칙 등 임포터의 정책을 입력

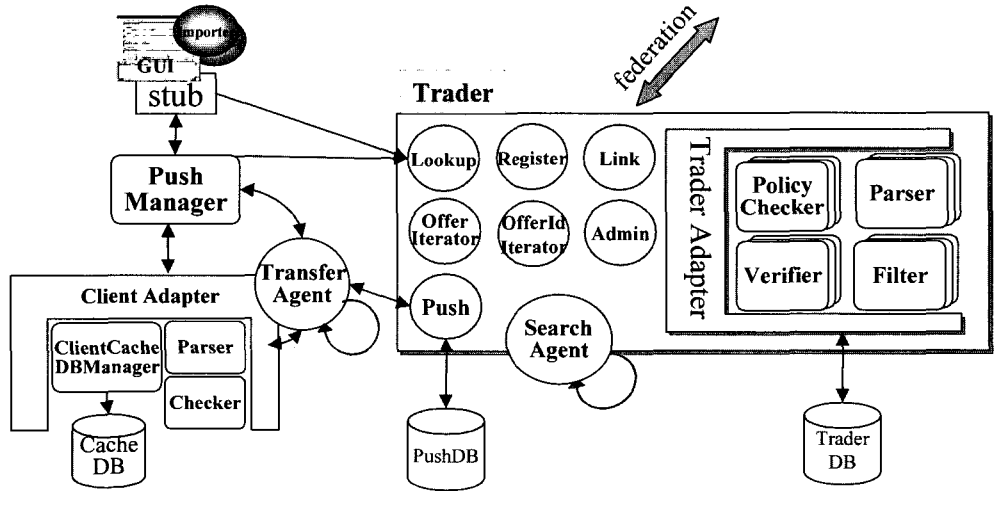

그림 5 PUTS의 구성도

할 수 있는 기능도 요구된다. 이러한 사항은 트레이더 서비스의 Lookup 인터페이스의 query() 메소드를 호출하기 위한 기본적인 매개변수에 해당하는 정보이다. 임포터는 질의에 대한 요청으로 전달받은 객체의 주소를 참조하여 정적 호출 또는 동적 호출을 수행하게 된다. 동적 호출시 객체의 인터페이스를 얻어 요청 객체를 동적으로 생성하여 호출한다. 이처럼 사용자가 트레이더 서비스의 이용을 용이하게 할 수 있도록 그래픽을 제공하는 모듈이 GUI 관리 모듈이다.

이러한 사항 외에 PUTS에서는 추가된 정책을 선택함으로써 푸시 여부를 결정할 수 있는 선택 기능을 지원한다. 또한, 사용자가 GUI없이 직접 코딩을 통해 위의 정보를 생성한 후 서비스를 요청할 수 있는 기능도 지원한다.

3.2.2 푸시 관리자 모듈

푸시 관리자 모듈은 클라이언트 상에서 매칭되는 서비스 오퍼를 검색하기 위한 기능을 제공한다. 이를 위해 Lookup 객체의 스텁에 있는 query() 메소드 내에서 GIOP 메시지로 마샬링되기 이전에 먼저 임포터 정책 및 푸시 여부를 판단할 수 있어야 한다. 즉, 임포터의 정책이 푸시를 수행하도록 설정되었다면 클라이언트는 지역 데이타베이스에서 매칭되는 서비스 오퍼가 있는지를 검색한다. 이 때 검색이 성공하면 바로 OfferIteratorHolder 객체에 데이타를 저장하여 전달한다. 검색이 실패하면 원래대로 원격 객체를 호출할 수 있도록 GIOP 메시지로 마샬링하여 전달하고, 전송 에이전트를 활성화시킨다. 하지만, 임포터의 정책이 푸시를 사용하지 않도록 설정되었다면 이러한 중간 과정 없이 곧바로 마샬링하여 원격 객체를 호출한다.

이처럼 지역 파일이나 데이타베이스를 검색하기 위해서 클라이언트로부터 질의를 받아 질의 정보를 파싱(Parsing)하고 필터링(Filtering)하며, 트레이더의 Lookup 인터페이스와 유사하게 검색된 데이타는 OfferIteratorHolder 객체를 통해 다시 스텁으로 전송하고 검색에 실패하면 'null' 값을 전송한다.

3.2.3 클라이언트 어댑터 모듈

트레이더에서 푸시된 서비스 정보는 클라이언트 캐쉬 데이타베이스에 저장된다. 캐쉬 데이타베이스에 저장된 정보는 서비스 유형별로 특성과 오퍼의 정보를 관리함으로써 질의어를 통해 검색될 수 있도록 구성된다.

클라이언트 어댑터는 푸시 관리자가 파싱한 정보를 바탕으로 데이타베이스 질의문을 작성하여 데이타베이스 관리자를 호출함으로써 데이타를 검색하는 역할과 전송 에이전트가 트레이더로부터 전송받은 서비스 오퍼를 캐쉬 데이타베이스에 저장하는 역할을 담당한다.

3.2.4 전송 에이전트 모듈

GUI나 임포터의 코드 내에 'use_push' 정책을 사용하여 서비스를 요청한 경우 먼저 클라이언트의 지역 데이타베이스를 검색한다. 이때 지역 데이타베이스에 매칭되는 서비스 오퍼가 존재하지 않는 경우 두 가지 작업이 병행되어 수행된다. 하나는 트레이더로 질의를 전송하여 푸시를 요청하는 작업이고 또 하나는 전송 에이전트가 활성화되어 트레이더의 검색 에이전트와 통신을 수행하는 작업이다. 전송 에이전트는 트레이더 상의 검색 에이전트가 검색이 완료되었음을 알려오면 검색된 데이타를 트레이더에서 클라이언트의 지역 데이타베이스로 전송한다.

3.2.5 트레이더 어댑터 모듈

클라이언트로부터 전달된 질의 정보 내에는 임포터가 설정한 정책, 서비스 유형, 제약 사항, 우선 조건 등이 포함되어 있다. 따라서 이러한 사항들을 해당 처리 모듈로 전이하는 모듈이 트레이더 어댑터 모듈이다.

트레이더 어댑터 모듈에는 먼저 임포터 및 트레이더의 정책을 비교하여 임포터 정책이 트레이더 정책 범위에 있는지를 판단하는 정책 검사기(Policy Checker), 제약 사항이나 우선 조건을 파싱하는 파서(Parser), 구문이나 문법적 오류를 체크하는 구문 분석기(Lexical Analyzer), 트레이더의 여러 데이타베이스와의 연동을 위한 데이타베이스 관리자(Database Manager) 등을 포함한다.

특히, 정책 검사기는 임포터 정책과 트레이더 정책을 판단하여 푸시 기능을 수행할 경우, 검색을 바로 수행하지 않고 질의 정보를 트레이더의 푸시 데이타베이스(PushDB)에 저장할 수 있도록 데이타베이스 관리자를 호출한다. 트레이더의 푸시 데이타베이스에 정보가 저장되면 검색 에이전트가 활성화되어 백그라운드로 검색을 실행한다.

3.2.6 검색 에이전트 모듈

검색 에이전트는 푸시 정책이 설정되었을 경우에 한해서 구동된다. 일반적으로 푸시를 설정하지 않은 경우는 트레이더가 실시간으로 검색을 수행하여 Holder객체에 담아 임포터에 넘겨주지만, 푸시를 설정한 경우는 푸시 데이타베이스에서 클라이언트로 푸시되지 않은 사항이 있는지를 검사한다. 또한 검색 에이전트는 푸시 데이타베이스에 저장된 정보를 바탕으로 지역 트레이더와 타 트레이더와의 연합에 따른 검색을 수행하고 검색이 완료되면 클라이언트의 전송 에이전트에 검색 완료를 알린다[14,15].

이러한 푸시 관련 모듈외에 연결형 트레이더의 기본적 기능을 제공하기 위해서는 Lookup, Register, OfferIterator, OfferIdIterator, Link, Admin 등의 인터페이스를 구현한 모듈이 트레이더에 존재해야 한다[2].

3.3 주요 모듈 설계 및 구현

PUTS는 OMG에서 정의한 기본적인 인터페이스를 지원하고 푸시 기능을 위한 인터페이스가 추가된다. 또한 서비스 검색 및 트레이더 연합을 위한 기능을 지원한다.

3.3.1 클라이언트 관련 모듈 설계

그림 6은 클라이언트 상에서 푸시 기능을 제공하기 위한 개략 수준의 클래스 다이어그램(Class Diagram)[13]이다.

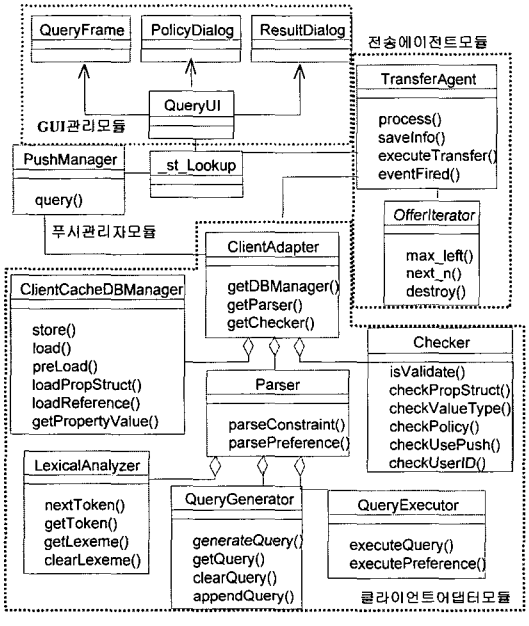

그림 6 클라이언트 관련 클래스 다이어그램

GUI 관리 모듈은 사용자에게 정책 설정 및 질의 작성을 용이하게 하고 검색 결과를 사용자에게 편리하게 보여주는 모듈로서 QueryUI 클래스와 QueryFrame, PolicyDialog, ResultDialog 클래스로 구성된다. 푸시 관리자 모듈은 클라이언트 상에서 지역 데이타베이스를 검색할 수 있도록 Lookup 인터페이스와 유사하게 query() 메소드를 제공하여 스텁에서 지역 데이타베이스를 검색하게 한다. 또한, 클라이언트와 트레이더간의 원격 호출을 위한 프락시 객체를 관리하며, _st_InterfaceName의 형태의 클래스를 관리한다. 전송 에이전트 모듈은 트레이더로부터 검색된 서비스 오퍼를 전송받기 위한 TransferAgent 클래스와 전송된 데이타를 검색하기 위한 OfferIterator 클래스로 구성된다.

ClientAdapter 클래스는 클라이언트 상에서의 데이타베이스 검색을 위한 ClientCacheDBManager 클래스와 구문 분석 및 오류 체크를 위한 Parser와 Checker클래스로 구성된다. 특히 Parser클래스는 구문 분석을 위한 LexicalAnalyzer 클래스와 데이타베이스 질의어 작성을 위한 QueryGenerator, QueryExecutor 클래스로 각각 구성된다.

그림 7은 클라이언트에서의 질의를 요청한 경우의 처리 순서를 나타낸다.

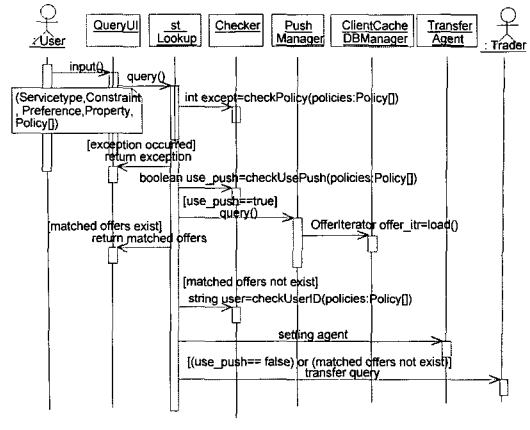

그림 7 클라이언트 상에서의 처리 순서

3.3.2 서버 관련 모듈 설계

PUTS를 지원하는 트레이더 상에는 크게 연결형 트레이더의 기본적인 인터페이스와 이의 구현 클래스, 푸시 기능을 지원하기 위한 클래스, 기타 클래스 군으로 구분된다. 그림 8은 트레이더 서비스를 제공하는 서버 상에서 푸시 기능을 제공하기 위한 PUTS의 관련 클래스의 관계를 나타낸 클래스 다이어그램이다.

클라이언트가 질의를 전송하면 전송된 질의 정보를 파싱하고 에러 체크 및 정책 체크를 수행하게 되는데 이때 이러한 기능을 수행하는 클래스가 트레이더 어댑터 모듈의 Parser, PolicyChecker, Verifier 클래스이다. 또한 Filter 클래스는 검색된 데이타를 정렬하거나 중복 데이타를 제거하는 역할을 담당한다.

푸시 처리 모듈의 PushDBManager 클래스는 임포터가 푸시 정책을 수행하도록 설정한 경우에 한해 요청한 서비스 정보를 저장하고, SearchAgent 클래스는 푸

시데이타베이스에 새로 추가된 데이타가 존재할 경우 검색 정보를 읽어 검색을 실행한다. 이때 타 트레이더와의 연합을 위해 Link클래스에서 타 트레이더의 주소를 참조한다. 지역 또는 연합을 통해 검색을 완료하면 검색 에이전트는 클라이언트의 전송 에이전트에 검색 완료 이벤트를 전달하고 전송 에이전트는 데이타 전송을 실행한다.

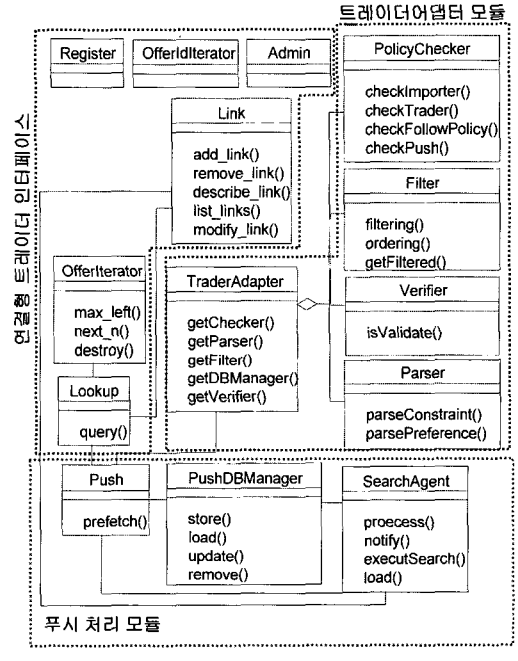

그림 8 서버 관련 모듈 클래스 다이어그램

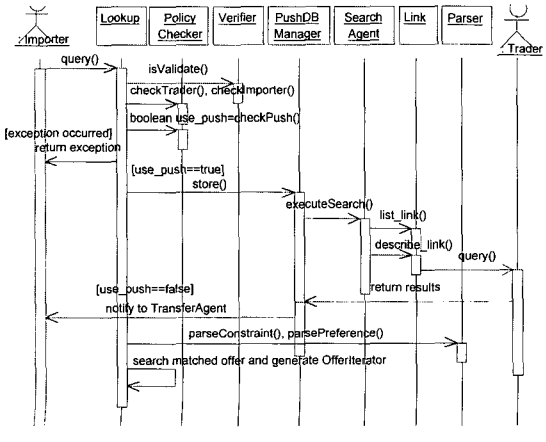

그림 9 서버 상에서의 처리 순서

그림 9는 서버 상의 객체간 메시지 흐름을 나타낸 순차도이다. 우선, PolicyChecker 객체는 전송받은 질의 정보를 검사하여 임포터의 푸시 여부를 판단한다. 푸시를 설정한 경우에는 정보를 푸시 데이타베이스에 저장하고, 검색 에이전트가 데이타베이스의 갱신 내용을 읽어 타 트레이더와 연합을 수행하거나 지역 트레이더 상에서 매칭되는 서비스 오퍼를 찾는다.

3.3.3 Push 인터페이스 구현

그림 10은 Push인터페이스에 대한 IDL(Interface Definition Language)[1]을 나타낸다. Push 인터페이스 내에는 클라이언트가 푸시를 요청한 경우, 트레이더가 검색을 완료하면 검색 정보를 트레이더에서 클라이언트

```
interface Push{
typedef Istring UserName;
attribute boolean transferred;
void prefetch(in UserName name, in ServiceTypeName type,
              out OfferSeq offers, out OfferIterator offer_itr)
 raise(NotImplemented, IllegalServiceType, UnknownServiceType);
};
```

그림 10 Push 인터페이스

```
public class PushImpl extends _PushImplBase{
        ..............
public void prefetch(String user,String type,
                     OfferSeqHolder offers,
                     OfferIteratorHolder offer_itr){
  throws NotImplement,UnknownServiceType,IllegalServiceType;
  PushDBManager dbmanager = TraderAdapter.getDBManager();
  if(!(user.length()==0)){
  Hashtable lists = dbmanager.load(user);
        :
  OfferSeqHolder offers; OfferIteratorHolder offer_itr;
  Vector offer_list=new Vector();
  OfferAdapter o_adapter = new OfferAdapter();
  PropertyAdapter p_adapter = new PropertyAdapter();
  ParsingAdapter parsing = new ParsingAdapter(type,constr,pref);
        :
  Offer[] searched_offer = new Offer[count];
  for(int i=0;i<count;i++){
    searched_offer[i]= new Offer(
      o_adapter.loadReference((String)offer_list.elementAt(i)),
      p_adapter.load((String)offer_list.elementAt(i)));
  }
  dbmanager.update (user_id,type);
  offers.value = searched_offer;
  OfferIterator itr_impl=newOfferIteratorImpl(searched_offer);
  _boa().obj_is_ready(itr_impl);
  offer_itr.value = itr_impl;      ..............
```

그림 11 Push 인터페이스 구현 코드

의 캐쉬 데이타베이스로 전송받기 위한 prefetch() 메소드를 정의한다. prefetch() 메소드는 사용자의 아이디와 서비스 유형 정보를 트레이더로 전송하고, 트레이더는 검색 에이전트가 검색한 데이타를 OfferSeqHolder 객체와 OfferIteratorHolder 객체에 담아 클라이언트로 전송한다. 이 때 클라이언트 상에서 트레이더의 Push 인터페이스를 접근하는 객체는 전송 에이전트가 된다.

그림 11은 위 IDL을 실제로 구현한 PushImpl 클래스의 개략적인 구현 코드를 나타낸다.

3.3.4 전송 에이전트 구현

전송 에이전트는 클라이언트가 푸시로 서비스를 요청한 경우에 한해, 백그라운드로 수행된다. 트레이더 서버의 Push인터페이스의 prefetch() 메소드를 주기적으로 호출함으로써 요청한 서비스가 검색되었는지를 판단한다. 검색 에이전트가 검색되었음을 알리면 전송 에이전트는 클라이언트의 캐쉬 데이타베이스로 데이타를 전송받는다.

그림 12는 이러한 기능을 수행하는 전송 에이전트의 구현 코드를 개략적으로 제시한 것이다.

```
public class TransferAgent extends CIAgent {
  public TransferAgent(String Name){
    super(Name);
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
   Push push=PushHelper.bind(orb,"Push");
  }
  public void run() {
    while(stopped == false){
      Thread.sleep((long)interval) ; // in milliseconds
      push.prefetch(userName,serviceType, offers, offer_itr);
      PushDBManager manager = ClientAdapter.getDBManager();
      OfferIterator itr=offer_itr.value;
      do{
       OfferSeqHolder offers_holder = new OfferSeqHolder();
       more_offers = itr.next_n(1,offers_holder);
       Offer offer = offers_holder.value[0];
       Property[] props = offer.properties;
       manager.store(ORB.init().object_to_string(offer.reference),
                     props,getServiceType());
      }while(more_offers);
      stop();
  }}
```

그림 12 전송 에이전트 구현 코드

3.3.5 푸시 관리자 모듈 구현

푸시 관리자 모듈의 핵심은 클라이언트에서 요청한 서비스 객체를 찾기 위해 원격으로 메시지가 전송되기 이전에 매칭되는 객체의 주소가 클라이언트 데이타베이스에 존재하는지 찾아 전달하는 것이다.

그림 13은 이를 위한 Lookup인터페이스의 스텁 코드 일부를 수정한 것이다. 코드에서 강조된 부분이 스트림(Stream) 형태의 GIOP 메시지로 전송되기 이전에 임포터의 정책 내용을 판단하여 처리하는 부분이다. 푸시로 설정한 경우는 PushManager 객체가 클라이언트의 지역 데이타베이스를 검색하여 매칭되는 정보를 찾고, 검색이 성공하면 곧바로 클라이언트의 인터페이스에 검색된 결과를 출력한다.

```
public class _st_Lookup extends _st_TraderComponents
                        implements CosTrading.Lookup {
                    :
public void query( java.lang.String type,.....)
 throws   CosTrading.IllegalServiceType,...{
                    :
  Checker checker = ClientAdapter.getChecker();
  int except=checker.checkPolicy(policiees);
  switch(except){ .....}
  boolean use_push = checker.checkUsePush(policies);
  if(use_push==true){
    PushManager manager = new PushManager();
    manager.query(type,constr,pref,policies,desired_props,
                  how_many, offers,offer_itr,limits_applied);
    if(offer_itr.value != null) return;  // exist in local Cache DB
    else{  // create Transfer Agent
     TransferAgent tagent = new TransferAgent("Transfer");
     tagent.setUserName(checker.checkUserID(policies));
        : // setting Transfer Agent
    }
  if(use_push != true || offer_itr.value==null){
    while(true) {
     _output = this._request("query", true); ..........
```

그림 13 푸시 기능을 위한 스텁 코드

4. PUTS 시스템 구현 및 성능 평가

본 장에서는 PUTS 모델을 적용하여 구현한 시스템을 제시하고, PUTS 시스템 성능을 여러 조건하에서 실험, 평가한다.

4.1 구현 환경

PUTS시스템은 인프라이즈(Inprise)사의 자바용 비지브로커(VisiBroker for Java)[10,11]라는 ORB제품을 이용하여 개발하였다. 자바용 비지브로커의 버전은 3.3이며, 구현 언어는 Java 1.1이다. 서버와 클라이언트에서 데이타를 저장하고 검색 및 삭제를 위해 마이크로소프트(Microsoft)사의 엑세스(Access)라는 관계형 데이타베이스를 사용하였으며, Java와 관계형 데이타베이스의 연동을 위해 Java Database Connectivity(JDBC)

2.0[16]을 이용하였다.

4.2 PUTS 시스템 구현

PUTS는 OMG의 트레이더 서비스 스펙에 따라 구현하였다. 스펙에 제시한 임포터 및 트레이더, 링크의 정책을 지원하며, 푸시 기능을 위해 기존의 정책에 새로운 정책을 추가하여 구현하였다.

타 트레이더와의 연합을 수행하기 위해 트레이더의 여러 형태 중에서 Lookup, Register, Admin, Link, OfferIterator, OfferIdIterator 인터페이스를 지원하는 연결형 트레이더로 구현하였다. PUTS는 클라이언트 상에 지역 데이타베이스를 구축하여 트레이더로부터 전송되는 많은 서비스 오퍼를 저장, 관리할 수 있으며, 트레이더와 클라이언트 상에서 각각 검색 에이전트와 전송 에이전트가 구동한다.

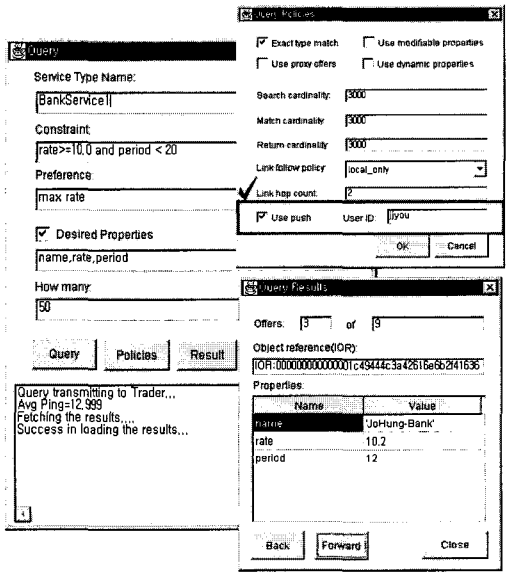

그림 14 클라이언트의 서비스 요청 및 서비스 오퍼 검색

그림 14의 QueryPolicies 프레임은 사용자가 서비스를 요청할 때 정책 설정을 위한 박스이다. 임포터의 정책 설정 항목은 OMG에서 제시한 기본적인 임포터의 정책과 PUTS 모델에서 제시한 푸시 정책이 있다. 정책 설정 박스의 하단 부분에 표시된 부분이 PUTS 시스템의 정책 설정 항목이다. 그 외의 9가지 항목은 OMG에서 정의한 임포터의 정책을 설정하기 위한 항목들이다. 클라이언트가 PUTS를 이용하여 서비스 오퍼를 푸시 형태로 전송받고자 할 경우에는 'Use-push'항목을 체크하고 'User-ID'필드에 사용자의 이름이나 아이디(ID)를 입력한다. 사용자가 입력한 정보는 이 후 서비스를 요청할 때 서버로 전송되고 서버의 데이타베이스에 저장 및 관리된다. 하지만 사용자가 'Use-push'항목을 체크하지 않은 경우는 푸시를 실행하지 않고 일반적인 트레이더 서비스처럼 실시간으로 서비스 오퍼를 전송받게 된다.

임포터의 정책을 설정한 후, Query 프레임에 검색하고자 하는 서비스 오퍼에 대한 질의 정보를 입력한다. 질의 정보는 서비스 유형, 제약 사항, 우선 조건, 전송받고자 하는 특성의 종류, 전송받고자 하는 서비스 오퍼의 수이다. 그림에서 'Constraint' 필드는 많은 트레이더가 관리하는 많은 서비스 오퍼 중에서 요구에 일치하는 서비스 오퍼를 전송받기 위한 제약 사항으로, 서비스에서 제공하는 특성 이름과 특성 값 등을 연산자와 함께 입력한다. 이때 입력할 수 있는 기본 사항은 OMG에서 정의한 제약 사항 언어 문법(Constraint Language BNF)을 따른다. 'Preference' 필드는 검색되어 전송되는 서비스 오퍼의 순서를 결정하는 우선 순위 항목이다. 필드에 'max rate'로 입력하면 'rate'라는 특성 이름을 가진 서비스 오퍼 중에서 특성 값이 가장 큰 순으로 전송하게 된다. 'Desired Properties' 항목은 서비스 오퍼에 있는 여러 특성 중에서 전송받고자 하는 특성들만을 입력한다.

하지만, 이렇게 많은 정보를 입력함으로써 사용자나 트레이더 관리자에게 부담으로 작용할 수 있다. 따라서, PUTS에서는 제약 사항이나, 우선 조건, 특성의 종류와 같은 여러 입력 정보를 입력하지 않고, 서비스 유형만을 입력하여 질의를 수행할 수 있다. 이러한 경우, 임포터의 정책은 기본값으로 설정되어 질의의 매개 변수로 입력된다. 즉, 사용자는 원하는 서비스 유형만으로 질의를 수행할 수도 있으며, 다양한 정보를 입력함으로써 좀더 정확하게 질의를 수행할 수도 있다.

트레이더의 데이타베이스나 클라이언트의 지역 데이타베이스에서 매칭되는 서비스 오퍼를 찾으면 QueryResults 프레임을 통해서 서비스 오퍼의 주소(IOR)와 특성의 이름, 값을 검색할 수 있다. 그림 14의 우측 하단 프레임은 클라이언트로 전송된 서비스 오퍼를 사용자가 검색하는 박스이다. 사용자는 전송된 서비스 오퍼 정보를 검색하여 가장 적합한 서비스 오퍼를 선택한 후 주소를 참조하여 서비스를 받게 된다.

PUTS 시스템에서는 임포터의 정책 설정 상태가 'use_push'인 경우 클라이언트의 지역 데이타베이스를 검색하고, 검색된 데이타가 없는 경우 질의를 트레이더로 전송한 후 클라이언트 상에서 전송 에이전트를 활성

화 시킨다. 전송 에이전트는 백그라운드로 수행되면서 트레이더로부터 검색 완료 메시지를 기다린다.

그림 15의 (B)는 활성화된 전송 에이전트가 트레이더의 검색 에이전트로부터 검색 완료 메시지를 받은 후 데이타를 지역상의 캐쉬 데이타베이스로 전송받는 과정을 나타낸다. 전송이 성공적으로 수행되면 전송 에이전트는 비활성화된다. (A)는 트레이더 서비스를 제공하는 서버의 실행과정을 나타낸다. 서버는 클라이언트의 정책에 따라 실시간으로 서비스를 제공하거나 검색 에이전트를 백그라운드로 실행시키면서 지역 트레이더나 타 트레이더와의 연합을 통해 검색을 수행한다. 푸시형으로 검색을 실행하는 경우 검색 에이전트는 검색 완료 메시지를 전송 에이전트에 전송한다.

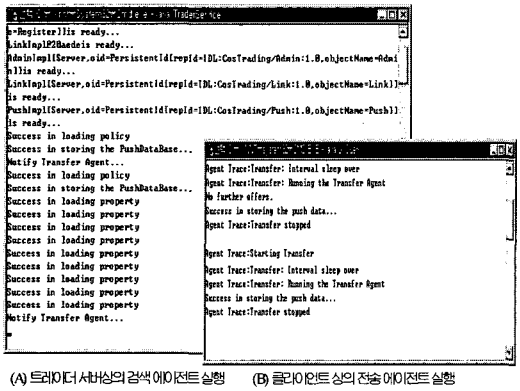

그림 15 PUTS 시스템의 구동 에이전트

4.3 성능 평가

PUTS의 성능 평가를 위해 그림 16과 같이 실험 환경을 구성하였다. 임포터와 엑스포터가 존재하는 LAN상에 지역 트레이더가 3개 존재한다. 지역 트레이더A는 가상의 금융 기관의 서비스 객체를 관리하고 있으며, 관리 객체에는 연동가능 주소, 금융기관 이름, 이자율, 전화 번호가 포함되어 있다. 지역 트레이더B는 금융 기관 및 호텔 관련 서비스 객체를 관리하고 있으며, 금융기관 정보는 트레이더A가 관리하는 내용과 같으며, 호텔 관련 서비스 객체에는 호텔의 이름, 가격, 전화번호, 주소 정보가 포함되어 있다. 지역 트레이더C는 임포터의 시작 트레이더로서 지역 트레이더A와 같이 금융 기관 관련 서비스 객체를 관리한다. 원격 트레이더A는 지역 트레이더C나 A처럼 금융기관 관련 정보를 관리하고 있으며, 원격 트레이더B는 지역 트레이더B처럼 금융기관 및 호텔 관련 정보를 관리한다. 각 트레이더의 관리 객체의 수는 지역 트레이더A와 C, 원격 트레이더A가 각각 1000개의 객체를 관리하며, 지역 트레이더B와 원격 트레이더B는 금융기관 관련 정보로 500개, 호텔 관련 정보로 500개를 관리하고 있다.

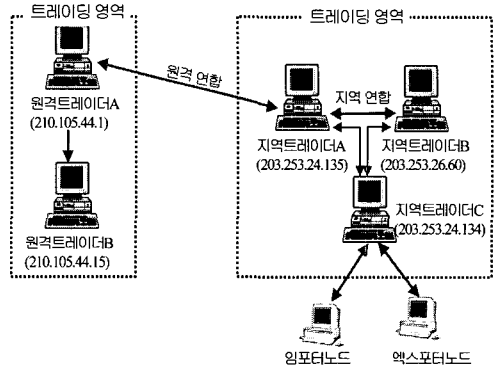

그림 16 실험 환경 구성도

지역 트레이딩 영역의 각 트레이더는 같은 LAN상에 존재하며 네트워크는 Ethernet환경의 T1전용선으로 구성되고, 원격 트레이딩 영역은 ISDN 56Kbps로 구성된다. 지역 트레이딩 영역의 트레이더의 하드웨어 및 소프트웨어 환경은 지역 트레이더A와 C가 RAM 128MB, Pentium-166MHZ의 CPU, WindowNT 4.0의 운영체제로 구성되고, 지역 트레이더 B는 RAM 64MB, Pentium-150MHZ, Windows98로 구성된다. 원격 트레이딩 영역의 각 트레이더 노드는 128MB의 RAM, PentiumII-333MHZ, WindowNT 4.0으로 구성하였다. 각 트레이더의 ORB 제품은 클라이언트와 서버 모두 자바용 비지브로커 3.2 버전을 사용하였다.

각 트레이더 연합의 대기 시간의 평균값을 측정하기 위해 전송되는 서비스 오퍼의 수를 50개로 고정시키고, 호출부터 데이타 전송까지의 회수를 매 100회로 하여 각 요청의 대기시간의 평균을 측정하였다. 그림 17은 일반적인 트레이더 서비스 기능을 사용하여 서비스 오퍼를 검색한 경우와 PUTS시스템을 사용하여 서비스 오퍼를 클라이언트의 지역 데이타베이스에서 검색한 경우의 사용자 대기시간을 비교한 그래프이다. 그래프의 가로축은 검색한 서비스 오퍼의 수이고 세로축은 사용자의 대기시간의 평균값이다. 그림에서 각 그래프는 PUTS의 전송에이전트가 모든 연합을 수행하여 데이타를 푸시하여 놓은 경우, 실시간으로 지역 트레이더C로부터 연합을 수행하지 않고 바로 서비스 오퍼를 전송

받은 경우, 같은 도메인 내에서 트레이더C와 A, B간에 연합을 수행하여 데이타를 검색한 경우, 다른 도메인 간의 트레이더간의 연합을 수행한 경우이다. 마지막 경우의 트레이더 연합은 지역 트레이더C, A, B와 원격 트레이더A, B가 연합을 수행하게 된다.

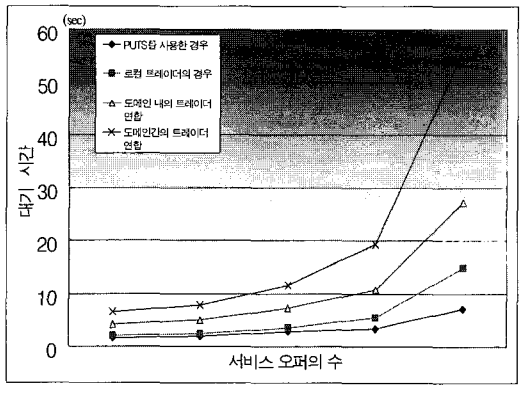

그림 17 트레이더 유형별 성능 평가

실험 결과를 통해 각 트레이더는 서비스 오퍼의 개수가 증가할수록 대기시간은 증가함을 알 수 있다. 지역 트레이더보다 같은 도메인에서 트레이더간 연합이 수행된 경우의 대기시간이 더 크며, 서로 다른 도메인에서의 트레이더간 연합이 수행된 경우의 대기시간이 가장 크다. 특히 트레이더간 연합이 수행될 경우 사용자의 대기시간은 연합의 수에 비례하며, 네트워크 트래픽이라는 시스템 외적 요소까지 고려하면 트래픽이 높은 시간대의 사용자 대기시간은 더욱 증가하게 된다.

이러한 트레이더 유형별 사용자 대기시간을 수식으로 표현하면 수식 1과 같이 나타낼 수 있다. *ARTt*는 전형적인 트레이더 서비스를 사용한 경우의 평균 대기 시간을 의미한다. 만약 '*hc*'가 0이면, 클라이언트는 연합을 수행하지 않고 서비스 오퍼를 전송받고자 하는 경우이다. 즉, 클라이언트는 지역 트레이더 중에서 시작 트레이더 내에서만 검색된 데이타를 전송받게 된다. '*hc*'가 0보다 큰 경우는 트레이더가 연합을 수행하여 검색한 데이타를 전송받게 된다. '*T*'는 데이타 전송 시간으로 클라이언트로부터 트레이더, 트레이더로부터 클라이언트, 트레이더간의 데이타 전송 시간을 포함한다. '*h*'는 적중률로서 클라이언트가 요청한 서비스 오퍼가 클라이언트의 지역 캐쉬 데이타베이스에 존재할 확률을 의미한다.

PUTS에서 푸시 정책을 사용하는 경우, 이러한 적중률이 대기시간을 결정하는 중요 요소가 된다. 만약 매칭

$$ART_t = \sum_{hc=0}^{n-1}(2T_{hc} + Rs_{hc}) + Os$$

$$ART_s = R_c \times h + \{ART_t \times (1-h)\}$$

ARTt=전형적인 트레이더 서비스의 평균 응답 시간
ARTs=PUTS를 사용한 경우의 평균 응답 시간
T=데이터 전송시간, *h*=적중률, *hc*= 연합의 수
Rs=서버 데이터베이스의 평균 검색시간
Rc=클라이언트 지역 캐쉬 데이터베이스의 평균 검색 시간
Os=서비스 오퍼의 평균 정렬 시간

수식 1 평균 대기 시간

되는 서비스 오퍼가 지역 캐쉬 데이타베이스에 존재하면 응답 시간은 향상되어 사용자 대기시간은 짧아지게 된다. 하지만, 지역 캐쉬 데이타베이스에 요청 데이타가 존재하지 않은 경우 클라이언트의 캐쉬 데이타베이스를 검색하는 시간이 추가되므로 오히려 응답 시간은 늘어나게 된다. 따라서, 향후 지역 캐쉬 데이타베이스의 데이타 적중률을 향상시킬 수 있는 연구가 요구된다.

PUTS시스템의 사용자는 일반적인 트레이더의 기능을 사용할 것인가, 아니면 푸시 형태의 서비스를 사용할 것인가를 설정하여 서비스를 요청하게 된다. 즉, PUTS 시스템은 일반적인 트레이더 서비스가 제공하는 모든 기능을 지원함과 동시에 푸시 기능을 제공한다.

표 3 일반 트레이더 시스템과 PUTS 시스템의 비교

유형 / 평가 요소	일반 트레이더 서비스 시스템	PUTS 시스템
푸시 기능	지원하지 않음	지원함
오프라인 검색	지원하지 않음	지원함
대기시간 요소	연합, 서비스오퍼 수, 트래픽 상태	서비스 오퍼 수, 매칭 확률
인터페이스	OMG 표준	OMG 표준 + 추가
임포터/트레이더 정책	OMG 표준	OMG 표준 + 추가
클라이언트 데이타베이스	필요 없음	필요함

표 3은 일반적인 트레이더 서비스 시스템과 PUTS시스템의 성능을 비교한 것이다. PUTS시스템은 푸시를 통해서 사용자가 원하는 정보를 미리 클라이언트의 지역 데이타베이스에 전송받아 검색할 수 있으므로 일단 푸시된 데이타는 오프라인(Off-line)으로도 검색할 수 있으며 대기시간을 최소화할 수 있다. 하지만, 데이타를

클라이언트 상에 저장하기 위해 클라이언트 데이타베이스가 구축되어야 가능하므로 클라이언트의 부하가 커질 수 있다는 단점이 있다.

5. 결론 및 향후 연구과제

트레이더 서비스에서 서비스 오퍼의 개수와 트레이더 간 연합에 따른 사용자 대기시간의 문제는 검색 엔진을 사용하여 데이타를 검색하는 경우와 유사하다. 검색 엔진의 성능 평가 기준이 검색 결과의 정확성과 검색 속도에 있듯이 트레이더 서비스의 성능도 같은 기준이 적용된다. 검색의 정확성은 사용자가 정책이나 제약 사항, 우선 조건 등을 상세하게 입력함으로써 어느 정도 해결할 수 있으나 검색 속도와 이에 따른 대기시간의 문제는 일반적인 트레이더 서비스의 구조로는 해결하기가 쉽지 않다. 특히, 한 트레이더가 다른 도메인의 트레이더와 많은 연합을 수행할 경우 매칭되는 서비스 오퍼가 클라이언트까지 전송되기까지는 네트워크의 트래픽, 트레이더 연합의 수, 전송될 서비스 오퍼의 양에 따라 대기시간은 증가하게 된다.

본 논문에서는 이러한 검색 속도와 대기시간의 문제를 PUTS모델로 해결할 수 있음을 제안하였다. PUTS 모델에서는 사용자가 원하는 서비스를 미리 푸시받을 수 있도록 임포터와 트레이더에 새로운 정책 및 인터페이스가 추가되어야 한다. PUTS모델을 적용한 PUTS시스템은 기존의 트레이더 서비스의 모든 기능을 지원하면서 동시에 푸시 기능을 제공한다. 따라서, PUTS시스템을 이용할 경우 사용자가 요구하는 바에 따라 서비스를 제공할 수 있으며 사용자의 대기시간도 단축시킬 수 있다.

향후 연구는 트레이더 서비스의 성능을 평가하는 또 다른 기준인 검색의 정확성 측면에서 서비스 오퍼를 정확하게 검색할 수 있는 모델의 연구가 필요하다. 또한 트레이더 연합시 최대한 빠른 검색을 수행하기 위한 알고리즘의 연구가 필요하다.

참 고 문 헌

[1] CORBA Specification v2.2, OMG Document Feb, 1998.

[2] CORBAservices: Common Object Services Specification, OMG Document 97-12-02.

[3] IONA CORBA Trader, http://www-usa.iona.com/news/pressroom/trader.html, IONA Technologies, 1998.

[4] Object Oriented Concepts, Inc, http://www.ooc.com/trader/, ORBacus Trader, 1998.

[5] Bearman, M., Raymond, K., Federating Traders: An ODP Adventure, Proceedings of the IFIP TC6/WG6.4, International Workshop on Open Distributed Processing, Berlin, Germany, 8-11 October, 1991.

[6] Bearman, M., Trading in Open Distributed Environment, CRC for Distributed System Technology, Univ. of Canberra, Feb., Brisbane, 1995.

[7] 박성원, 안순신, 트레이더 간 연합을 위한 트레이딩 영역 분할 모델 및 성능 분석, 정보과학회지, 제25권, 제9호, pp.937~953, 1998.

[8] Burger, C., Cooperation Policies for Traders, Proceedings of the International Conference on Open Distributed Processing, February 1995, Brisbane, Australia.

[9] Vogel, A., Enabling Interworking of Traders, Proceedings of the International Conference on Open Distributed Processing, February 1995, Brisbane, Australia.

[10] Inprise,VisiBroker for Java: Programmers Guide, v3.3, Inprise Corporation,1998.

[11] Pedrick, D., and others, Programming with VisiBroker: A Developers Guide to VisiBroker for Java, John Wiley & Sons, Inc., 1998.

[12] Orfali.R, Client/Server Programming with JAVA and CORBA, 2nd, John Wiley & Sons, Inc., 1998.

[13] Fowler, M., UML distilled: Applying the standard object modeling language, Addison-Wesley, May, 1997.

[14] Joseph, P. B., Bigus, J., Constructing Intelligent Agent with Java: A Programmers Guide to Smarter Application, John Wiley & Sons, Inc., 1997.

[15] Watson,M., Intelligent Java Applications for the internet and Intranets, Morgan Kaufmann Publishers, San Francisco, California, 1997.

[16] Haecke,B.V., JDBC:Java Database Connectivity, IDG Books Worldwide, Inc.,1997.

유 재 정

1994년 서울교육대학교 초등교육학과 교육학사. 1997년 숭실대학교 전자계산학과 공학사. 1999년 숭실대학교 컴퓨터학과 공학석사. 1999년 7월 ~ 현재 서울중랑초등학교 교사. 관심분야는 컴퓨터교육, 전자상거래, 분산 객체 컴퓨팅

윤 범 렬

1997년 경원대학교 전자계산학과 공학사. 1998년 ~ 현재 숭실대학교 전자계산학과 석사과정. 오브젝소프트(주) 선임연구원. 관심분야는 분산객체 컴퓨팅, 전자상거래 시스템, 이동 에이전트

김 수 동

1984년 미조리 주립대학교 전산학과 졸업(학사). 1988년 The University of Iowa, 전산학 석사. 1991년 The University of Iowa, 전산학 박사. 1991년 ~ 1993년 한국통신 연구개발단 선임연구원. 1994년 현대전자 소프트웨어연구소 책임연구원. 1995년 ~ 현재 숭실대학교 컴퓨터학부 조교수. 관심분야는 객체지향 개발방법론, 분산객체 컴퓨팅, 전자상거래 시스템.