그리드 환경에서 메타서비스 기반의 워크플로우 시스템

(A Workflow System based on Meta-Services in Grid Environments)

이 진 복 † 이 상 근 † 최 재 영 †† 변 옥 환 †††
(Jinbock Lee) (Sangkeon Lee) (Jaeyoung Choi) (Okhwan Byeon)

요 약 본 논문에서는 그리드 환경에서 워크플로우 형태의 작업을 효율적으로 관리할 수 있는 워크플로우 시스템을 소개하고자 한다. 이 시스템에서는 워크플로우를 메타서비스 개념으로 구성하여 재사용성을 높였고, 사용자는 서비스 호출만으로 다양한 형태의 서비스를 실행할 수 있다. 또한 사용자는 GUI 형태의 편집기를 이용하여 워크플로우를 작성하거나 편집하는데 편리성을 제공받을 수 있다. 그리고 워크플로우의 재사용성과 확장성을 높이기 위하여 워크플로우 모델을 서비스, 플로우, 태스크와 같이 3개의 계층으로 분할하여 구성하였다. 이로써 본 논문에서 제안하는 미들웨어는 사용자 편리성과 함께 그리드 자원을 최대한 효율적으로 이용하도록 제공해준다.

키워드 : 그리드, 메타서비스, 워크플로우, MSF

Abstract In this paper, we present a workflow system which manages efficiently operations in workflow form. The workflow of this system is made up of concepts of meta-services for increasing reusability. Thus users can execute the workflow by calling their services. Also, GUI workflow editor is developed with the workflow system which provides users with convenience. Furthermore the workflow model is divided into 3-layers such as service, flow, and task layer for reusability and scalability of workflow. Therefore, this middleware can use grid resources effectively and offer convenience to users.

Key words : GRID, Metaservices, Workflow, MSF

1. 서 론

과거에는 로컬 컴퓨터의 자원만을 이용하여 작업을 처리하였지만, 네트워크 기술의 발달되면서 네트워크상에 연결된 수많은 컴퓨터들의 자원을 동시에 이용할 수 있게 되었다. 1990년대 초반에 그리드 컴퓨팅[1]이 소개되면서 파일 형태의 데이타에서부터 CPU, 메모리 등 기타 하드웨어 장비에 이르기까지 모든 자원을 공유하여 사용할 수 있게 되었다. 이러한 공유 자원들을 운영체제나 미들웨어에서 통합하여 효율적으로 관리한다면 사용자는 그리드 환경의 자원에 대해서 신경쓰지 않아도 된다. 즉, 사용자나 응용 프로그램에서는 사용할 그리드 자원의 위치가 어디인지, 어떠한 형태로 되어있는지 등의 상태를 몰라도 원하는 작업을 처리할 수 있다[2].

단일 작업이 아닌 여러 프로세스들로 복잡하게 구성되거나 반복적으로 수행되는 작업들을 하나의 흐름으로 기술한 것이 워크플로우이다. 이러한 워크플로우를 분산된 환경에서 명세하고 실행, 모니터링 등의 기능으로 구성되어 있는 것이 워크플로우 시스템이다[3]. 또한 그리드 환경에서 작업의 목표를 성취하기 위하여 미리 정의된 프로세스들의 정보와 작업 흐름에 관련된 절차를 자동화하는 방법이 필요하게 되었다. 프로세스의 자동화 모델을 정의하기 위한 메타 모델로써의 워크플로우는 목표 달성을 위한 프로세스들 간의 순차적 혹은 동시에 수행되는 작업들의 흐름이나 그 시스템으로 정의된다.

그리드 컴퓨팅은 대규모로 진행되는 과학 연구에서 필요한 높은 성능을 컴퓨팅 자원의 통합과 지능적인 관

† 학생회원 : 숭실대학교 컴퓨터학과
jeinbi@ssu.ac.kr
seventy9@ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
choi@ssu.ac.kr
††† 종신회원 : 한국과학기술정보연구원 e-Science
ohbyeon@kisti.re.kr
논문접수 : 2007년 6월 13일
심사완료 : 2007년 7월 22일

리를 통해서 제공한다. 그러나 성공적인 과학 연구 환경을 제공하기 위해서는 컴퓨팅 자원의 관리뿐만 아니라 과학 연구에서 사용되는 응용프로그램과 데이타, 그리고 이것들을 통합하는 워크플로우 등도 컴퓨팅 환경에 통합되어야 한다.

그리드 기반 사이언스 포탈의 요구사항들을 통합하여 그리드 컴퓨팅에 기반한 과학 연구 포탈을 구축하면 그림 1과 같은 계층 구조를 가진다. 컴퓨팅 자원들은 Globus Toolkit[4]과 같은 인프라 구축 도구들로 통합되며, 통합된 보안, 원격 작업 제출, 데이타 교환 등 컴퓨팅 자원들이 통합될 수 있도록 개별적인 자원에 접근하는데 필요한 인터페이스를 제공한다. 자원 관리자 계층은 상위 계층에서 요청한 작업들을 하위 계층에서 제공하는 컴퓨팅 자원들에 분배하고 실행시키는 역할을 한다. 응용 로직 계층에서는 사용자 작업을 위한 응용프로그램, 데이타 및 그와 관련된 메타데이타들을 관리하고, 사용자 인터페이스를 통해 호출된 사용자 서비스와 파라미터들을 시스템에 저장된 메타데이타들을 활용해서 자원 관리자 계층에서 실행이 가능한 작업으로 변환하는 역할을 한다. 웹 서버 계층은 사용자의 서비스를 호출하는 웹 어플리케이션들을 관리하며, 사용자 인터페이스는 사용자의 요청을 웹 어플리케이션에 전달한다. 이러한 계층 구조 중에서 사용자 인터페이스/웹서버 계층은 아파치/톰캣 웹서버와 PHP, JSP/Servlet, Portlet 등의 기술을 활용해서 효율적으로 구축할 수 있다. 그리고 기존에 구축된 많은 그리드 시스템들과 자원을 공유하기 위해서는 컴퓨팅 자원들에 Globus Toolkit을 설치하여야 하며 대부분의 플랫폼에 설치 가능하므로, Globus Toolkit을 컴퓨팅 자원들에 설치하는 것으로 컴퓨팅 자원 계층과 그리드 인프라 미들웨어 계층을 구축할 수 있다. 따라서 자원 관리 계층과 응용 관리 계층을 통합하여 프레임워크를 구성하면 복잡한 사이언스 포탈 개발을 단순화할 수 있으며, 프레임워크를 구성하는 다양한 모듈 중 필요한 모듈의 설정을 변경하거나 해당 모듈만을 추가로 구현하여 손쉽게 그리드 기반의 사이언스 포탈을 구축할 수 있다.

본 논문에서 제안하는 프레임워크는 사이언스 포탈

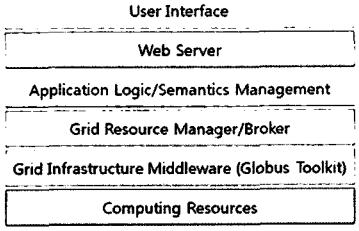

그림 1 그리드 기반 사이언스 포탈의 계층 구조

구축을 위한 것이므로, 사이언스 포탈을 구축하는데 필요한 워크플로우를 중심으로 모델링하였다. 워크플로우를 외부에서 접근하기 위한 인터페이스를 정의하는 계층, 워크플로우의 제어 패스와 데이타 패스를 정의하는 계층, 워크플로우에서 호출되는 태스크를 정의하는 3개의 계층으로 분할하였다. 그리고 각각의 계층을 처리하는 모듈들에 해당 계층의 자원 관리를 위한 작업을 정의함으로써 각각의 규칙을 정의하고, 인증 모듈과, 각각의 계층에서 사용하는 XML 명세를 저장 관리할 수 있는 모듈을 분할하여 크게 다섯 개의 모듈을 가지는 구조로 정의하였다. 또한 새로운 워크플로우를 작성하거나 기존의 워크플로우를 재구성할 때, 사용자는 워크플로우 편집기를 이용하여 편리성을 제공받을 수 있다.

2. 관련연구

그리드 환경을 구축하기 위한 기본적인 도구로 가장 널리 사용되는 것은 Globus Toolkit[5]이다. Globus Toolkit은 컴퓨팅 자원들을 공유하기 위해 필요한 기본적인 인증 서비스, 작업 실행, 데이타 전송, 시스템 정보 통합 등의 기능을 제공한다. 하지만 Globus Toolkit에서는 워크플로우 처리에 대한 기능을 제공하지 않기 때문에, 그리드 환경에서 작업을 제출할 수 있는 워크플로우 시스템이 필요하게 되었다. 또한 Grid Lab의 GAT (Grid Application Toolkit)[6]는 그리드 미들웨어들의 기능을 서비스화하여, API 호출을 통해 미들웨어 기능들을 검색하고 호출하는 기능을 제공한다. 하지만 그리드 미들웨어를 모듈화하여 API 없이 동적으로 프레임워크에 통합하기 위한 기술이 필요하고, 손쉽게 설정하고 관리할 수 있는 구성 요소의 개발이 필요하다.

워크플로우를 구성하는 작업들 간의 관계를 나타내는 워크플로우 패턴을 표준화하기 위한 대표적인 연구 사례로는 XPDL[7], YAWL[8,9] 등이 있다. WfMC(Workflow Management Coalition)[10]에서 진행 중인 XPDL (XML Process Definition Language) 프로젝트는 워크플로우를 표준화하여 XML 스키마로 정의하려는 연구이다. XPDL은 작업이 실행되는 다양한 환경의 메타데이타들을 모아 공통된 부분을 추출하여 작업 흐름간의 인터페이스 연결 부분을 XML 스키마로 정의한다. 따라서 XPDL을 기반으로 개발된 워크플로우 시스템간에 연동도 가능하다. 그러나 XPDL은 여러 작업 환경들의 메타데이타들을 반영하기 위한 것이어서 워크플로우 모델이 매우 복잡하고 방대하므로 구현하기가 어렵다.

워크플로우 패턴에 관한 또 다른 연구 사례로는 YAWL(Yet Another Workflow Language) 시스템이 있다. YAWL 시스템은 Petri Nets[11]을 기반으로 보다 다양한 워크플로우 패턴을 지원할 수 있도록 확장한

것이다. YAWL은 워크플로우 엔진과 작업을 실행하는 브로커를 분리하여 브로커 부분만 구현하면 새로운 형태의 작업을 워크플로우에 통합할 수 있다. 그러나 YAWL 시스템은 웹서비스를 위한 브로커는 지원하지만 기존 응용 프로그램을 실행하기 위한 브로커가 지원하지 않는다. 또한 일반 사용자가 사용하기에는 사용법이 어렵게 되어있다.

또한 분산 컴퓨팅 환경에서 작업을 효과적으로 수행하기 위한 워크플로우로 대표적인 것으로는 DAG_MAN [12]이 있다. DAG_MAN은 분산 컴퓨팅 환경에서 효율적인 워크플로우 관리 기능을 제공하기 위해서 워크플로우에 의존적인 자원들을 스케줄링하는 기능을 제공한다. 하지만 작업들을 재사용할 수 있는 기능을 제공하지 않는다.

본 논문에서 소개하는 워크플로우 시스템은 HG2C (Human Gene to Chemicals)[13] 연구를 위한 그리드 컴퓨팅 미들웨어이다. HG2C 연구는 그리드 기술을 기반으로 신약 개발을 위해 필요한 서열 상동성 검색, 단백질 구조 모델 생성, 신약 후보 물질 도출 등 신약 연구에 필요한 작업들을 시나리오로 통합하고, 이 시나리오를 다수의 화합물질을 대상으로 적용할 수 있는 신약 연구 환경을 제공하는 것을 목표로 하고 있다. 이러한 시나리오의 작성과 수정 및 재사용을 위해서는 워크플로우 환경이 제공되어야 하며, 다수의 화합물질을 대상으로 빠른 결과를 도출하기 위해서는 그리드 컴퓨팅 기술을 이용한 자원의 통합이 필수적이다. 따라서 이 두 가지 모델을 통합하여 워크플로우 모델과 컴퓨팅 자원 모델 각각의 특성을 유지하면서 효율적으로 동작하는 미들웨어가 필요하다.

BT 분야 중 신약 연구는 대부분 다양한 소프트웨어와 데이타들을 통합하여 사용자가 복합적인 연구를 수행할 수 있는 환경을 제공하는 것을 목표로 한다. HG2C와 유사한 국외의 연구로는 미국 샌디애고 슈퍼컴퓨팅센터(SDSC)의 EOL(Encyclopedia of Life) 프로젝트[14]가 있다. EOL 프로젝트는 모든 종의 모든 단백질 정보를 공공의 이익을 위해서 수집하고 색인화하는 것을 목표로 하며, 단백질 정보를 수집하는 여러 소프트웨어를 단계적으로 연결하여 파이프라인을 구성하는 단순화된 워크플로우 시스템이다. EOL 프로젝트 이외에도 e-Science 환경을 제공하기 위한 대표적인 PSE(Problem Solving Environment)로는 Cactus[15]와 Triana [16]을 들 수 있다. Cactus는 과학 시뮬레이션을 위한 코드를 모듈로 만들어 프레임워크에 통합하는 기능을 제공하며, Triana는 워크플로우로 여러 과학 실험을 연결해서 사용할 수 있는 환경을 제공한다. 그러나 이러한 연구 환경만으로는 그리드 컴퓨팅 자원 상에서 작업을 스케줄링할 수 없기 때문에 스케줄러나 워크플로우 시스템 등을 함께 사용하여 e-Science 환경을 구축한다.

3. 워크플로우 모델

3.1 메타서비스에 기반한 워크플로우 모델

워크플로우를 서비스 지향구조에 통합하는 일은 워크플로우를 구성하는 태스크의 재사용성을 높이는 일이라기보다는 워크플로우 로직 자체를 서비스로 제공할 수 있도록 하는 일이다. 그리고 워크플로우 로직을 서비스로 제공하는 것은 서비스에 접근할 수 있는 서비스 인터페이스를 정의하고, 전달된 서비스 파라미터 값에 따라 변경된 워크플로우 로직이 수행되는 것을 의미한다.

기존의 워크플로우 재사용성에 관한 연구들은 워크플로우를 구성하는 태스크를 객체지향적으로 설계하거나, 태스크들을 연결한 소규모의 워크플로우를 하나의 모듈처럼 사용할 수 있도록 통합하는 방법을 사용하여 주로 태스크의 재사용성을 높이는 연구가 진행되었다. 또 템플릿을 이용하여 옵션에 따른 워크플로우를 생성하는 방법으로 워크플로우 로직을 정의하였다.

그러나 사이언스 포탈에서 사용되는 워크플로우 로직을 서비스로 제공하기 위해서는 모듈의 재사용성이나 템플릿 기능 보다는 IDL(Interface Description Language)을 정의하는 것이 효율적이다. 서비스 인터페이스와 로직을 분리하는 아이디어의 유용성은 이미 다양한 컴포넌트 모델에서 사용하는 IDL, CORBA, 웹서비스 등의 기술에서 이미 입증되었다. 소프트웨어 재사용을 위한 아키텍쳐인 CORBA에서는 IDL과 ORB라는 개념의 브로커를 활용하여 기존의 레거시 언어들로 작성된 모듈을 서비스 지향구조에 통합하여 객체로 서비스하였다. 이러한 구조는 이후에 등장한 웹서비스 기술에서도 WSDL과 웹서비스 컨테이너로 계승되었다. 메타서비스[17]는 이러한 개념을 워크플로우를 서비스 지향 구조에 통합하는 방법에도 적용하여, 메타서비스 마크업 언어와 브로커로 워크플로우를 서비스하는 방법을 제공한다.

3.2 메타서비스에 기반한 워크플로우 모델의 효율성

워크플로우에서 재사용성을 높이기 위한 방안으로 많은 워크플로우에서 태스크를 하드웨어 IC 모듈의 서비스 재사용성을 위한 기본 메타포로 잡고 있다. IC 모듈이 외부와 통신하는 각각의 핀은 서비스 파라미터를 제공하는 것으로 볼 수 있다. 그리고 핀에서 입력되는 신호들은 코어에 위치한 로직을 거쳐서 신호 출력을 담당하는 핀으로 출력된다. 즉, 동일한 로직을 가지는 코어가 다른 핀수를 가지는 칩으로 패키징될 수 있다.

동일한 로직을 상이한 인터페이스로 표현하는 예는 C/C++, Java 등의 객체 지향 언어에서 다형성이라는 개념으로 표현된다. 이 경우 메소드의 이름은 동일하고 파

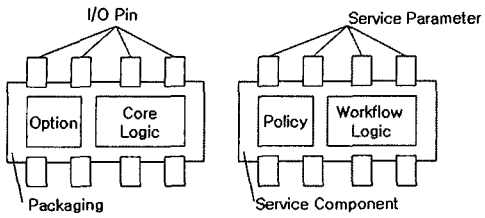

그림 2 하드웨어 IC 모듈의 서비스 재사용성

라미터의 수가 다른데, 실제 코드는 이 메소드 중 가장 파라미터가 많은 메소드에서 구현하고, 파라미터의 수가 적은 메소드는 가장 파라미터가 많은 메소드 중 몇 개의 파라미터를 원하는 값으로 고정하는 경우가 많다.

이렇게 워크플로우가 반복적으로 재사용되는 경우에는 제어 패스나 데이타 패스가 변화하여 재사용되기보다는 서비스 파라미터와 연결된 몇몇 부분만 변화한다. 이렇게 변화하는 부분을 서비스 파라미터 값으로 대체할 수 있는 인터페이스를 선언한다면 기존의 워크플로우 엔진을 새로 개발하지 않고 인터페이스 처리 모듈만을 추가함으로써, 워크플로우를 서비스로 선언하고 재사용할 수 있다. 또 메타서비스는 기존에 파싱된 워크플로우 로직을 재사용할 수 있으므로 워크플로우 시스템의 성능을 향상시킬 수 있다.

3.3 메타서비스 기반의 워크플로우 예제

본 장에서는 메타서비스를 설명하기 위한 워크플로우의 예를 살펴보고자 한다. 우선 아래 그림 3과 같은 워크플로우에서 t1~t6의 단위 작업이 DAG(Direct Acyclic Graph) 구조로 연결되어 있으며, 워크플로우의 입력은 키보드로 받고 출력은 디스플레이를 이용한다. 하지만 입력이나 출력을 파일로 변경하고자 하는 경우에, 사용자는 기존에 작성된 워크플로우를 복사하여 해당 속성을 변경한 후 저장하거나 원본을 수정하여야 한다. 이로 인해서 원본을 수정하는 경우 원본이 보존되지 않는 문제가 발생할 수 있으며, 원본을 복사하는 경우에도 근

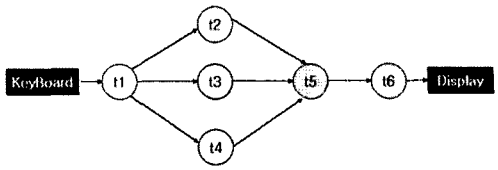

그림 3 워크플로우 예

본적인 기능은 동일한 워크플로우가 중복되어 저장된다. 따라서 메타서비스의 개념으로 구성하면 이와 같은 문제점을 해결하면서 재사용성을 극대화시킬 수 있다.

위의 워크플로우를 A라고 할 때, 아래 그림 4와 같이 A를 wa라는 이름을 가지는 워크플로우 인스턴스화하여 포함할 수 있다. 또한 입력을 p1, 출력을 p2로 연결하여 워크플로우의 중복이나 재구성 없이 재사용성을 극대화하여 서비스를 생성할 수 있다.

```
MetaService M1.(p1, p2)
begin
    import Workflow A as wa;
    wa.t1.in = p1;
    wa.t6.out = p2;
end
```

그림 4 메타서비스 Pseudo Code 예제 1

```
MetaService M2 (p1)
begin
    import Workflow A as wa;
    wa.t1.in = p1;
end
```

그림 5 메타서비스 Pseudo Code 예제 2

위의 그림 5는 입력의 인자만 받아서 변경하는 코드의 예이다. 그림 4와 그림 5와 같이 메타서비스를 사용하면 하나의 워크플로우를 다른 형태의 서비스로 손쉽게 재사용할 수 있음을 알 수 있으며, 재사용된 워크플로우를 도식화하면 아래 그림 6과 같다.

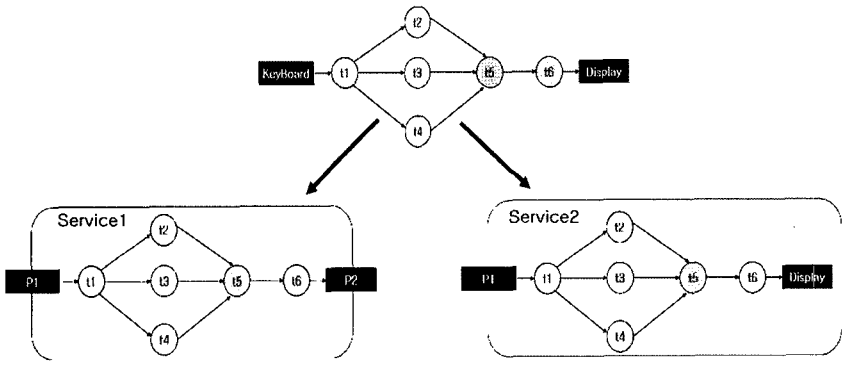

그림 6 메타서비스를 이용한 워크플로우의 재사용

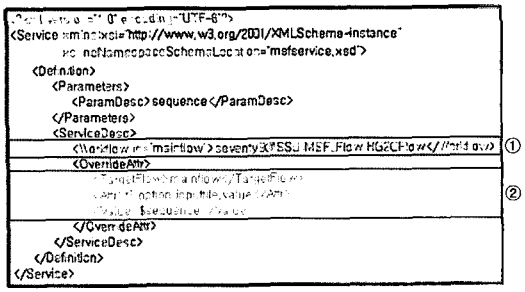

그림 7 워크플로우를 호출하는 서비스의 예

위의 그림 7은 MSF(Meta Service Framework)에서 실질적으로 사용되는 워크플로우 서비스의 설정을 XML 문서 파일로 표현한 것이다. ①과 같이 ID가 mainflow인 워크플로우 인스턴스를 선언한 다음에 ②와 같이 mainflow의 t1 태스크 중 inputfile 옵션 값을 입력되는 파라미터로 치환하여 워크플로우를 실행하라는 의미이다.

4. 메타서비스 프레임워크

본 장에서는 그리드 자원을 관리하는 계층부터 사용자에게 워크플로우 인터페이스를 제공하는 계층을 통합하여 그리드 포탈을 구축할 수 있는 미들웨어인 MSF (Meta Service Framework)에 대해 설명한다. MSF는 서비스 지향적인 워크플로우를 기반으로 워크플로우의 재사용성을 증대시키며, 기능에 따라 다섯 개의 에이전트로 구성되어 그리드 환경에 따라 최적화할 수 있다[18].

4.1 3계층 워크플로우 모델

MSF의 워크플로우 모델을 3개의 계층으로 분할하여 워크플로우의 확장 및 수정이 용이하므로 뛰어난 재사용성과 편의성을 제공한다. 이 3계층은 워크플로우를 서비스로 변환하기 위한 서비스 계층, 워크플로우를 구성하는 프로그램간의 관계와 데이타 흐름을 정의한 플로우 계층, 워크플로우를 구성하는 단위가 되는 프로그램들에 대한 정보를 기술한 태스크 계층으로 구성되어 있다.

4.1.1 서비스 계층

워크플로우가 사용하는 데이타 파일이나 프로그램에 전달되는 파라미터를 변경하는 등의 반복적으로 수행되는 작업을 효율적으로 처리하기 위한 방법과 정책을 서비스 계층에 기술한다. 메타서비스는 사용자가 서비스를 호출할 때 사용한 파라미터에 따라 사용자의 요구사항을 반영한 워크플로우가 동적으로 생성될 수 있도록 서비스 인터페이스와 워크플로우를 연결하는 방법과 정책을 기술한 중간 단계의 서비스이며, 워크플로우를 서비스의 형태로 호출하기 위한 서비스 인터페이스를 선언하고, 서비스 인자들의 값에 따라 워크플로우에 기술된 작업, 데이타 등의 속성 값이 변경되도록 매핑하는 역할을 한다.

4.1.2 플로우 계층

MSF의 플로우 계층에는 워크플로우를 구성하는 작업들의 관계를 선언하게 된다. 즉, 각 프로세스들 간의 작업 순서나 관계를 정의하여 아래 그림 8과 같은 DAG(Directed Acyclic Graph) 형태의 작업 흐름을 나타낼 수 있다.

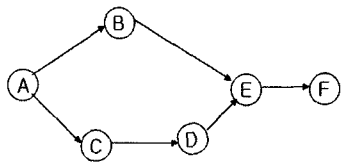

그림 8 Directed Acyclic Graph

4.1.3 태스크 계층

태스크 계층에서는 워크플로우의 각 프로세스 수행에 필요한 정보와 작업노드에 설치된 프로그램을 식별하기 위한 정보가 기술된다. 즉, 프로그램 실행 파일 이름이나 버전, 환경변수, 경로 및 파라미터 등을 정의한다.

4.2 시스템 구조

MSF 시스템은 아래 그림 9와 같이 다섯 개의 에이전트로 구성되어 있으며, 모두 자바로 구현되어 있다. 각 에이전트의 역할과 기능을 살펴보면, AM(Access Manager)은 전체 시스템의 인증을 담당하며, OM(Ontology Manager)은 서비스, 플로우, 태스크 계층을 기술하는 XML 파일들을 저장하고 관리한다. SM(Service Manager)은 사용자가 OM에 접속해서 서비스를 검색하고 호출하면 해당 서비스의 메타서비스 정보를 이용해서 워크플로우를 생성하는 기능을 가지고 있다. RM(Resource Manager)은 생성된 워크플로우를 해석하여 태스크로 분할하고 미리 수집된 자원 정보를 이용해서 스케줄링하며, EM(Execution Manager)에 할당하게 된다. EM에서는 할당받은 작업을 실행, 모니터링 등의 제어 기능을 하게 된다.

이와 같이 MSF를 다섯 개의 에이전트로 구성하면 다양하게 변화되는 그리드 환경에 최적화된 미들웨어를 구축할 수 있게 되며, 이로써 적응력이 뛰어나고 높은 유연성을 제공하게 된다.

4.3 워크플로우 스케줄링

MSF 시스템의 RM에서 사용자의 워크플로우를 스케줄링하기 위한 큐의 구조는 그림 10과 같다. 워크플로우 큐의 최상단에 사용자 큐가 있고, 이 사용자 큐는 현재 MSF 시스템에 작업을 제출한 사용자들 중에서 우선순위가 높은 사용자를 선택하는 역할을 한다. 또한 RM

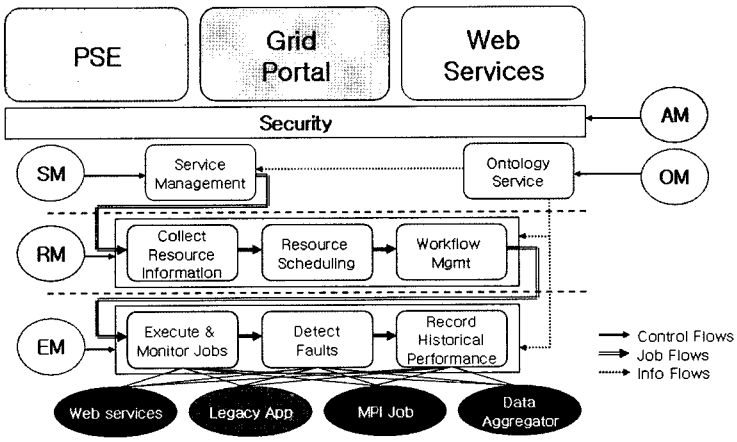

그림 9 MSF 시스템의 구조

은 사용자 별로 워크플로우 큐를 분리하여 관리하는데, 사용자가 선택되면 해당 사용자의 워크플로우 큐에서 작업이 스케줄링된다. 워크플로우 큐에는 워크플로우를 구성하는 태스크가 모두 완료된 워크플로우, 일부가 완료되어 현재 스케줄링해야 하는 작업이 남아있는 워크플로우, 아직 스케줄링되지 않은 워크플로우가 순서대로 정렬되어 있다. 태스크가 모두 완료된 워크플로우는 일정 수가 되면 워크플로우 큐에서 제거되며, 일부 작업만 완료되어 계속 수행이 필요한 워크플로우는 RM에 연결된 EM들이 동시에 수행할 수 있는 작업 수에 따라 관련된다.

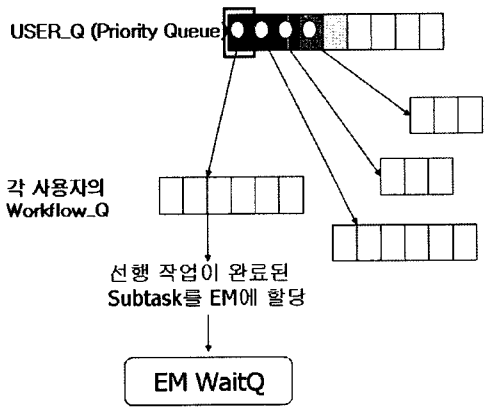

그림 10 RM의 워크플로우 스케줄링

4.4 워크플로우 편집기

MSF의 워크플로우 정보는 XML 문서 형태로 구성되어 있으며, 사용자가 이를 편리하게 관리 및 수정하기 위한 방법이 필요하게 되었다. 따라서 GUI 형태의 워크플로우 편집기를 제작하게 되었으며, 프로그램의 모듈

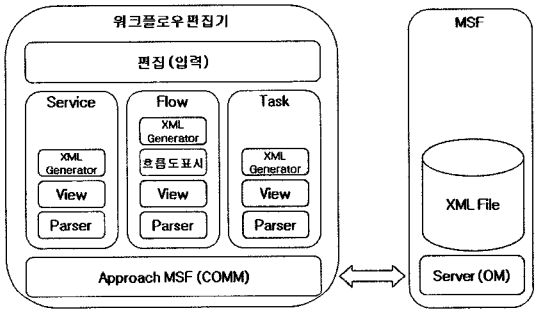

그림 11 워크플로우 편집기의 모듈 구조

구조는 그림 11과 같다. 편집기는 MSF 3계층의 정보가 기술되어 있는 XML 문서 파일을 관리하는 OM과의 접속으로 모든 데이타를 처리한다. OM으로부터 서비스, 플로우, 태스크 계층의 XML 문서를 전송받고, 이를 수정하여 다시 서버에 저장하는 클라이언트-서버 구조로 구현되어 있다. 워크플로우 편집기는 크게 다섯 가지의 모듈로 구성되어 있는데, 서버인 MSF 시스템과의 연결과 데이타 전송을 담당하는 통신 모듈과 사용자로부터 정보를 편집하기 위한 데이타를 입력받는 모듈, 그리고 각 3계층의 정보를 관리하는 모듈로 구성되어 있다[19].

편집기의 실행 흐름을 살펴보면, 프로그램을 실행하고 로그인과 동시에 서비스, 플로우, 태스크 3계층의 파일의 목록을 사용자에게 보여주고, 이 목록을 이용하여 사용자는 수정이 필요한 파일을 선택 후에 편집할 수 있다. 편집후에는 XML 문서를 재구성하여 OM을 통하여 서버에 저장되며, 이때 사용자가 직접 XML 문서 파일의 원본 내용을 확인할 수도 있다.

이러한 GUI 환경의 편집기를 이용함으로써 사용자는 3계층으로 구성된 워크플로우의 다양한 정보를 편리하고 효율적으로 관리할 수 있다. 또한 편집기로 편집된

워크플로우 정보는 MSF 시스템에 곧바로 적용되어 자동적으로 워크플로우의 내용이 변경된다. 이로써 워크플로우 정보는 사용자가 직관적으로 확인하면서 쉽게 편집이 가능하며, 시스템의 재시작이나 특별한 방법 없이 자동화된 미들웨어를 구축할 수 있다.

5. 시험 및 평가

5.1 시험

본 논문에서 소개하는 메타서비스 기반의 워크플로우 처리를 위한 미들웨어를 작동시키기 위해서는 AM, OM, SM, RM, EM과 같이 다섯 개의 에이전트를 실행해야 한다. 각 에이전트는 독립적으로 실행되며, TCP 통신을 통하여 이벤트 기반의 메시지를 서로 전송한다. 여기서 제안하는 미들웨어는 e-Science 워크플로우에 최적화되었지만, 각 에이전트는 독립적으로 작성되었기 때문에 다른 목적을 가지는 워크플로우 시스템에도 유용하게 사용할 수 있다. 즉, 에이전트의 변경이 필요하면 해당 에이전트의 수정만으로 다른 에이전트와의 유기적인 동작을 지속시킬 수 있다.

MSF 시스템의 실제 동작 과정을 살펴보면 다음과 같다. 아래 그림 12는 MSF 시스템의 기능을 시험하기 위한 샘플 코드이다. 코드를 실행하기 전에 AM에 로그인하여 컴퓨팅 자원에 접근하기 위한 프록시 인증서를 미리 생성하였고, 샘플코드는 SM에 접속하여 ①과 같이 seventy9@SSU:MSF_Service:HG2C 서비스의 sequence 파라미터에 aids.seq를 넘겨서 작업을 실행한다. 이 파라미터를 넘기는 코드는 아래 그림의 ②와 같으며, 이 코드에 의해 실행되는 서비스의 내용은 3.3장의 그림 7이다. seventy9@SSU:MSF_Flow:HG2CFlow 형태를 가지는 워크플로우 인스턴스를 선언한 다음 해당 플로우의 각 파라미터를 치환하여 워크플로우를 실행한다.

코드가 실행되면 서비스 요청을 받은 SM에서 해당

```
public static void main(String[] args) {
  MSFUser user = new MSFUser(TestSetting.id, TestSetting.pw);
  SMContext ctx = new SMContext("127.0.0.1", user);

  MSFDescriptor md
    = MSFDescriptor.decode("seventy9@SSU:MSF_Service:HG2C");

  /* MSFDescriptor md = new MSFDescriptor();
  md.setName("HG2C");
  md.setType(MSFDescriptor.MSF_Service);
  md.setVO("SSU");
  md.setUser("seventy9"); */

  ServiceDescriptor sd = new ServiceDescriptor(md);

  MSFServiceParam[] params
    = {new MSFServiceParam("sequence", "aids.seq")};

  ctx.requestService(sd, params);

  ctx.close();
}
```

그림 12 서비스 요청을 수행하는 자바 코드의 예

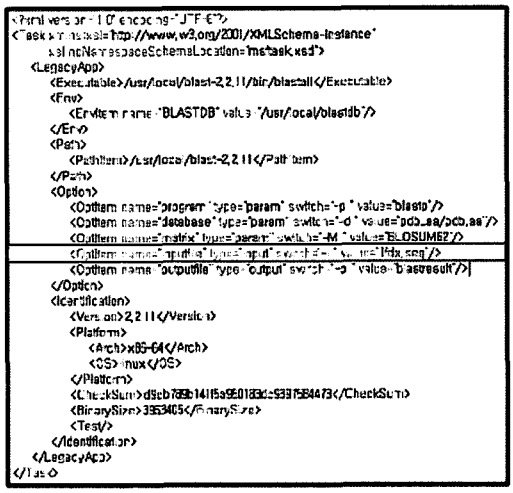

그림 13 태스크의 예

메타서비스가 호출되며, SM에서 해석된 서비스는 RM에서 워크플로우를 실행한다. 이때 워크플로우를 실행하기 위한 자료 구조를 생성하면서 교체할 옵션 값을 해당 태스크에 배분하고 정보를 분석하여 EM에 작업을 할당하게 된다. 그림 13은 EM에서 실행되는 태스크 중 blast의 예이다. 아래 내용은 ①과 같이 옵션 중에 input-file의 값이 1fdx.seq로 되어 있지만, 서비스를 호출할 때에 파라미터로 넘겨준 aids.seq로 변경되어 태스크가 실행된다. 따라서 태스크를 실행할 때에 이러한 옵션을 치환해 줌으로써, 워크플로우를 구성하는 태스크의 재사용성을 극대화할 수 있다.

EM에서 작업이 수행되면 각각의 작업을 Globus Toolkit의 GRAM에 제출하고 상태 정보를 태스크 큐의 자료 구조에 업데이트 한다. 또한 작업 수행 이전에 미리 준비되어야 하는 데이타가 있으면 RM에서 Grid-FTP로 전송하는데, 파일을 전송할 때에도 파라미터 치환 정보를 사용한다. 작업이 완료되면 각 자원별로 할당된 컴퓨팅 자원과 태스크 아이디 별로 구분된 작업 디렉토리 정보가 큐에 저장되어 있으므로, 이를 이용해서 결과 파일을 내려받을 수 있다.

그림 14는 위에서 설명한 서비스, 플로우, 태스크로 구성되어 있는 워크플로우 정보를 확인하고 수정하는 편집기의 실행 화면을 나타내고 있다. 이러한 GUI 환경의 워크플로우 편집기를 제공함으로써, 사용자에게 편리성을 제공하여 사용자는 직관적으로 워크플로우의 내용을 파악하고 수정할 수 있다. 특히 플로우 계층의 내용을 수정시에는 작업의 흐름을 도식적으로 표현하여 사용자는 워크플로우 처리 과정을 쉽게 인지할 수 있다.

실질적으로 사용자가 MSF 시스템을 사용할 경우에는 그림 15와 같이 웹 포탈로 접근하게 된다. 따라서

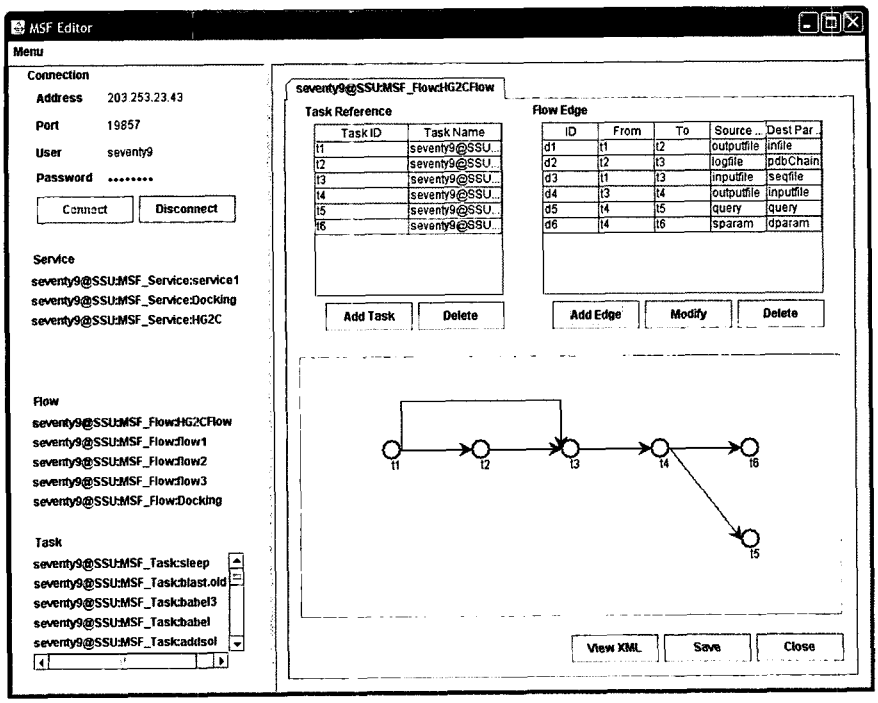

그림 14 워크플로우 편집기

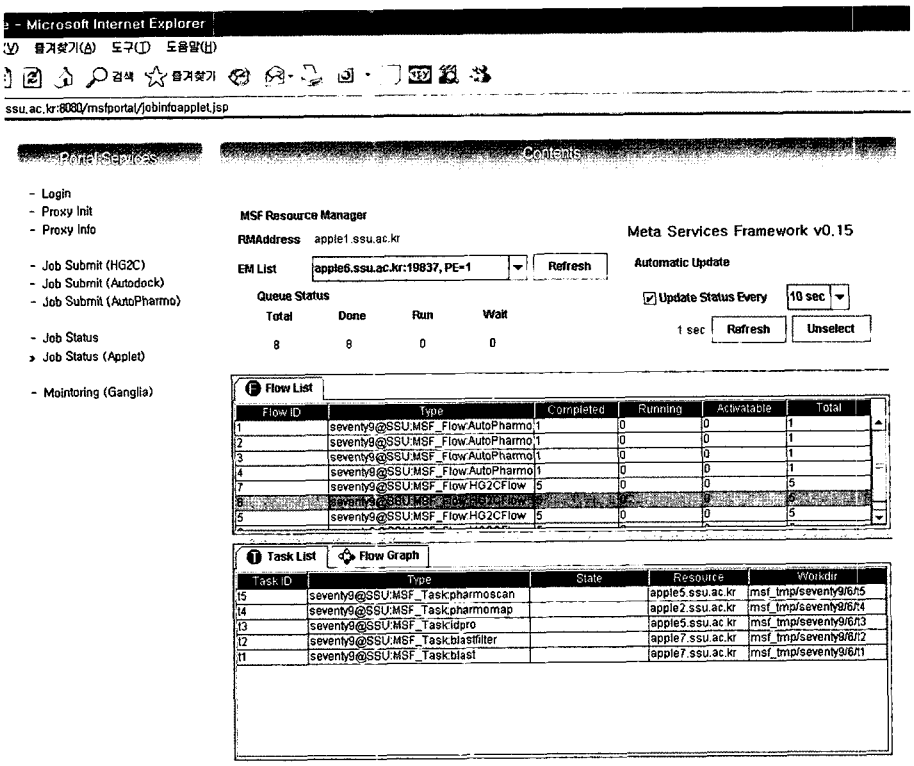

그림 15 MSF 웹포탈

워크플로우를 이용하는 사용자는 편리하게 작업을 제출하고 확인할 수 있으며, 클러스터 장비의 그리드 자원 사용 현황도 살펴볼 수 있다.

또한 MSF 시스템은 실행된 작업의 확인 및 디버깅을 위한 로깅 기능을 제공하고 있다. 로그 시스템의 구현에 널리 사용하는 log4j[20] 라이브러리를 활용해서 작업의 실행 상황을 파일로 로깅할 수 있다. 기본적으로 콘솔 화면과 로그 파일로 디버깅 정보와 사용 기록 로그가 저장되도록 설정되며, 로그 수준의 정보를 변경함으로써 로깅되는 내용을 조절할 수 있다.

5.2 평가

MSF 시스템에서 메타서비스 기반의 워크플로우를 수행시킨 결과, 기존 워크플로우의 높은 재사용성과 확장성을 확인할 수 있었다. 사용자가 서비스를 요청하면 워크플로우 모델의 서비스 계층에서 워크플로우와 태스크를 실행하고 파라미터를 치환하는 과정을 자동적으로 수행한다. 따라서 단순히 서비스 제공을 원하는 경우에는 서비스 이름과 파라미터 형식만 알면 사용자는 복잡한 워크플로우를 호출할 수 있다. 또한 새로운 서비스 인터페이스나 워크플로우를 작성하고 싶은 경우에는 GUI 형태의 편집기를 사용하여 직관적이고 편리하게 워크플로우를 재구성할 수 있다. 그리고 사용자는 MSF 시스템의 모든 기능을 웹 포탈을 통하여 접근함으로써, 특별한 작업환경이 필요하지 않고 웹으로 MSF 시스템을 이용할 수 있다.

6. 결론 및 향후 연구 계획

MSF 시스템은 사용자가 워크플로우를 작성하는 과정에서부터 그리드 환경에 작업을 제출하는 과정을 통합하여 그리드 포탈을 구축하고 관리하기 위한 노력을 줄여준다. 또한 기존의 그리드 환경을 위한 DAG 기반 워크플로우에 비해서 높은 재사용성을 제공하기 위해서 다음과 같은 특징을 가지도록 설계하였다. 워크플로우 모델을 서비스, 플로우, 태스크의 3계층으로 분할하여 워크플로우의 확장이 필요한 경우 해당 부분만 수정하면 되므로 뛰어난 확장성을 제공한다. 또한 서비스 계층에는 메타서비스 개념을 적용하여 워크플로우와 서비스 인터페이스를 연결하여 기존에 구성된 워크플로우의 재사용성을 극대화시킨다. 그리고 MSF 시스템의 모듈을 다섯 개의 에이전트로 구성하여 다양한 그리드 환경의 특성에 최적화하여 배치하고 설치할 수 있으므로 이전의 그리드 미들웨어들에 비해 높은 유연성을 제공한다. 마지막으로 새로운 워크플로우를 작성하거나 기존의 워크플로우를 재구성할 경우 GUI 형태의 편집기를 이용하여 쉽게 편집할 수 있는 환경을 제공한다.

현재 MSF 시스템은 많은 부분이 웹포탈과 같은 GUI 형태로 사용자에게 편리성을 제공하고 있다. 하지만 그리드 자원을 정의하는 MSF 시스템의 환경 설정 파일은 직접 XML 문서를 수정하여야 한다. 따라서 그리드 자원 추가와 같은 변경이 있을때, 이와 같은 환경 설정 파일을 웹을 통하여 관리자가 쉽게 수정할 수 있는 환경을 제공할 예정이다. 그리고 MSF 시스템을 MAGE (Mobile Agent based Grid Environment)[21] 시스템과 통합함으로써 모니터링 및 미들웨어의 원격관리가 가능하며 통계 정보를 제공할 수 있다. 또한 새로운 클러스터 장비가 추가되었을 경우에 MAGE를 이용하여 MSF 시스템의 자동설치 기능을 제공할 예정이다.

참 고 문 헌

[1] Ian Foster, Carl Kesselman, *"The Grid2: Blueprint for a New Computing Infrastructure,"* Morgan Kaufmann, 2004.

[2] George Coulouris, Jean Dollimore, Tim Kindberg, *"Distributed Systems: Concepts and Design - fourth edition,"* Addison Wesley, 2005.

[3] Jablonski, S. and C. Bussler, *"Workflow Management Systems: Modeling, Architecture, and Implementation,"* Thomson Press, 1996.

[4] Ian Foster, Carl Kesselman, *"Globus: A Metacomputing Infrastructure Toolkit,"* Intl J. Supercomputer Applications, 11(2):115-128, 1997.

[5] Globus Toolkit, *http://www.globus.org/toolkit*

[6] GridLab, *http://www.gridlab.org*

[7] WfMC, *"Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface - XML Process Definition Language(XPL) (WFMC-TC-1025). Technical report,"* Workflow Management Coalition, Lighthouse Point, Florida, USA, 2005.

[8] W.M.P. van der Aalst, *"The Application of Petri Nets to Workflow Management,"* The Journal of Circuits, Systems and Computers, 8(1):21-66, 1998.

[9] W.M.P van der Aalst and A.H.M ter Hofstede, "Design and Implementaion of the YAWL system," CAiSE 04, Riga, Latvia, Springer, June 2004.

[10] Workflow Management Coalition, *http://www.wfmc.org*

[11] W. Reisig, *"Petri Nets, An Introduction,"* EATCS, Monographs on Theoretical Computer Science, Springer Verlag, Berlin, 1985.

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, *"Condor-G: A Computation Management Agent for Multi-Institutional Grids,"* presented at 10th International Symposium on High Performance Distributed Computing, 2001.

[13] Human Genome to Chemicals for Drug Discovery(HG2C), http://www.hg2c.org

[14] EOL: The Encyclopedia of Life, http://eol.sdsc.edu

[15] T. Goodale, G. Allen, G. Lanfermann et al., *"The cactus framework and toolkit: Design and app-*

lications," In V. Hernandez J.M.L.M. Palma, J. Dongarra and A. A. Sousa, editors, Proceedings of VECPAR, 2002.
[16] Triana, *http://www.triana.co.uk*
[17] Sangkeon Lee, Jaeyoung Choi, *"Meta Services: Abstract a Workflow in Computational Grid Environments,"* LNCS 3516, ICCS 2005, pp.916-919, Springer, 2005.
[18] 이상근, 최재영, 이지수, "Meta Service Framework: e-Science 포탈 구축을 위한 에이전트 기반의 그리드 미들웨어 프레임워크", HPC 연구회, 2005.
[19] 이진복, 이상근, 최재영, "그리드 환경에서 3개의 계층으로 분리된 MSF를 위한 사용자 중심 워크플로우 편집기의 구현", 한국정보과학회 HPC연구회 제18권 제1호, pp.65-72, 2007.
[20] Logging Service, Apache Software Foundation, *http://logging.apache.org/log4j*
[21] 권성주, "동적 재구성이 가능한 다중 계층 구조 기반의 그리드 관리 시스템에 관한 연구," 숭실대학교 컴퓨터학부, 2006.

이 진 복
2007년 숭실대학교 컴퓨터학부 졸업(학사). 2007년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 분산처리, 그리드컴퓨팅

이 상 근
2001년 숭실대학교 컴퓨터학부(학사). 2003년 숭실대학교 컴퓨터학과(석사). 2003년~2007년 숭실대학교 컴퓨터학과 박사과정. 2007년~현재 (주)티맥스소프트 R&D 센터 선임연구원. 관심분야는 그리드컴퓨팅, 고성능컴퓨팅(HPC), 전자상거래

최 재 영
정보과학회논문지 : 시스템 및 이론
제 34 권 제 6 호 참조

변 옥 환
1979년 한국항공대학교 통신정보공학과 1993년 경희대학교 전자공학과(박사). 한국외국어대, 고려대학교, 경희대학교 겸임교수 역임. 미국 OSM Corp(1984), NCSA/UIUC(1997) 초청연구원. 1978년~1995년 KIST 시스템공학연구소 책임연구원, 연구전산망개발실장, 슈퍼컴퓨팅응용실장. 1995년~1999년 ETRI 슈퍼컴퓨팅센터 책임연구원, 고성능망연구실장, 슈퍼컴퓨팅연구실장. 1999년~현재 KISTI 초고속연구망부장, e-Science사업단장. 관심분야는 고성능망 관리 및 보안, 그리드네트워킹 및 협업시스템, e-Science 응용 및 미들웨어