그리드 환경에서 워크플로우 서비스를 제공하기 위한 메타 스케줄링 프레임워크

(A Meta Scheduling Framework for Workflow Service on the Grid)

황 석 찬 † 최 재 영 ††
(Seogchan Hwang) (Jaeyoung Choi)

요 약 그리드는 대규모의 독립된 자원을 공유하여 과학 연구와 같은 문제를 해결하기 위한 컴퓨팅 환경을 제공한다. Globus는 그리드를 구축할 수 있는 표준적인 미들웨어로서 자원 관리, 보안, 데이타, 정보 서비스 등의 핵심적인 서비스를 제공하지만 어플리케이션에서 필요한 다양한 요구를 충족시키기에는 아직 많은 연구가 필요하다. 그 중에서도 워크플로우 서비스는 복잡한 어플리케이션을 위한 중요한 서비스로 부각되고 있다. 본 논문에서는 워크플로우 서비스를 지원하는 메타 스케줄링 프레임워크 (MSF)를 제안한다. 메타 스케줄링 프레임워크는 그리드 작업에서 요구하는 내용과 작업 절차를 표현할 수 있는 XML 기반의 작업 제어 언어 (JCML)를 제공하며, 이를 바탕으로 복잡한 작업 처리를 효과적으로 관리할 수 있는 워크플로우 관리 서비스를 제공한다.

키워드 : 그리드, Globus, 워크플로우, 작업 제어 언어, 메타 스케줄링 프레임워크

Abstract The Grid is new infrastructure to provide computing environment for grand challenge research by sharing large-scale resources. Currently the Globus becomes a de facto standard middleware to construct Grid and supports core services such as resource management, security, data transfer, information services, and so on. However, it still needs more works and researches to satisfy requirements from various grid applications. A workflow management is becoming a main service as one of the important grid services for complex grid applications. We propose a Meta Scheduling Framework (MSF) in this paper. The MSF provides a XML-based Job Control Markup Language (JCML) for describing information and procedures of grid applications, and a workflow management service for scheduling the job using the JCML and for processing the job effectively.

Key words : Grid, Globus, Workflow, Job Control Markup Language, Meta Scheduling Framework

1. 서 론

그리드는 널리 분산되어 있는 이기종의 자원을 공유하여 현대의 도전적인 문제를 해결하는데 필수적인 컴퓨팅 환경으로 자리잡고 있다[1]. 그리드는 컴퓨팅 자원을 보유하고 있는 서로 다른 기관으로부터 형성된 동적인 가상 조직으로 컴퓨팅 자원을 공유하는 것뿐만 아니라 문제를 해결하는데 필요한 전문 연구자들의 협력 방법도 같이 지원하여 문제를 해결할 수 있는 환경을 제공한다. 그리드는 다양한 특성을 갖는 자원과 어플리케이션에서 요구하는 서비스를 연결해주기 위한 미들웨어의 역할이 매우 중요하다. Globus [2]는 그리드 구조의 미들웨어 계층을 담당하며 그리드를 구축하는 표준화된 도구로서 정보 서비스, 자원 관리, 데이타 전송, 보안 등의 기능을 제공한다 .

워크플로우 서비스는 비즈니스 프로세스에서 문서나 정보 또는 태스크들을 참여자들과의 작업 수행을 위해 미리 정해진 절차에 따라 전체 또는 일부분을 자동화하는 개념이다[3]. 이러한 워크플로우 서비스는 복잡한 처리 과정의 작업을 단일화된 방법으로 제어/관리할 수 있는 방법으로서, 그리드와 같이 다양하고 많은 자원을 이용하여 복잡한 계산 작업을 수행하는 환경에 필요한 부분으로 인식되고 있으며 그리드를 구성하는 여러 서비스들 중에서도 가장 중요한 서비스로 부각되고 있다. 예를 들어 여러 태스크로 구성되어 있는 어플리케이션의 경우 각 태스크의 특징에 맞는 자원에 할당되어 특정한 순서와 데이타의 배분을 필요로 하는 경우에 전체

† 비 회 원 : 한국과학기술정보연구원 연구원
seogchan@kisti.re.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
choi@comp.ssu.ac.kr
논문접수 : 2003년 10월 29일
심사완료 : 2004년 6월 21일

작업의 흐름을 관리할 수 있는 워크플로우 서비스를 필요로 한다.

Globus는 미들웨어로서 그리드의 핵심 서비스를 제공하지만 워크플로우 서비스와 같은 응용 어플리케이션에서 필요로 하는 다양한 서비스를 제공하지 못하고 있다. 또한 현재 개발자 중심의 Globus 이용 환경도 어플리케이션을 이용하는데 어려운 점으로 지적되고 있다. 이러한 부분에 대한 연구가 여러곳에서 응용 프로젝트에 맞추어서 다양한 형태로 진행되고 있다. 본 논문에서는 그리드에서 어플리케이션의 사용을 확대시킬 수 있는 메타 스케줄링 프레임워크(Meta Scheduling Framework, MSF)를 제안한다.

본 논문의 2장에서는 관련연구를 살펴보고 3장에서는 MSF의 전체적인 설계 방향과 각 구성 요소들을 상세하게 설명한다. 4장에서는 MSF의 구현과 실행 예제에 대해서 설명하며, 5장에서는 결론과 향후 연구방향에 대해서 논의한다.

2. 관련연구

지난 십여년동안의 워크플로우 기술과 관련한 연구는 비즈니스 프로세스 중심의 워크플로우 관리 시스템을 위한 표준화에 노력해 왔다. 이러한 워크플로우 기술을 그리드에 접목하기 위해 많은 연구자들이 다양한 방법으로 연구를 진행하고 있다. 그 중에서도 응용 프로젝트를 기반으로 해당 어플리케이션에서 필요로 하는 프로세스의 처리를 위한 워크플로우의 연구, 즉 해당 프로젝트를 위한 워크플로우의 연구와 그리드에서 적용할 수 있는 일반적인 워크플로우의 방법에 대한 제안으로 구분할 수 있다.

GridFlow[4]는 실행하는 어플리케이션의 성능을 예측하여 자원을 할당하는 방식으로 작업 흐름을 관리하는 시스템이다. 이 시스템은 하위 로컬 자원의 관리를 수행하는 에이전트(Titan)와 상위 글로벌 그리드 자원을 관리하는 ARMS를 이용하여 전체 그리드 자원의 성능을 모니터링하며 작업의 흐름을 제어한다. MyGrid[5]는 영국의 EPSRC에서 수행하고 있는 프로젝트로서 생물정보학을 지원하기 위해 그리드, 웹 서비스, 시멘틱 웹을 통합한 미들웨어 환경을 제공한다. 그리고 자원의 발견과 워크플로우 서비스, 분산 질의처리 등을 함께 제공한다. ASCI 그리드[6]는 글로버스를 이용한 CORBA 기반의 소프트웨어 구조를 사용하여 미국의 여러 국립연구소(LLNL, LANL, SNL)들을 연결한 그리드 시스템으로서 각 연구소의 서로 다른 자원과 소프트웨어 등 매우 복잡한 스크립트 등을 연결하기 위해 통일된 인터페이스와 관리를 위한 GALE(Grid Access Language for HPC Environemtn)을 이용하여 전체 시스템을 운영한다. Condor[7]는 컴퓨터의 유휴자원을 이용하는 HTC (High Throughput Computing) 기반의 분산 컴퓨팅 환경으로 고정된 형태의 자원을 이용하는 것이 아니라 사용되지 않는 시간의 컴퓨터를 유휴자원 풀(pool)로 구성하여 작업의 처리를 수행하고 있으며 작업내의 태스크들간의 의존관계와 같은 문제를 해결하기 위해 DAGMan 메타 스케줄러를 이용하여 작업의 흐름을 처리하고 있다. 각 프로젝트에서 사용되는 워크플로우의 연구 방법은 해당 프로젝트에서 목적으로 하는 연구 환경에 적용하기 위한 모델로서, GridFlow는 어플리케이션의 성능을 예측하여 스케줄링 하는 시스템으로 퍼지 타임기술을 이용하여 작업에 대한 최적의 자원할당을 목적으로 구성된 시스템을 기반으로 하고 있으며, MyGrid는 온톨로지를 이용하여 계속적으로 추가되는 데이타를 인식하고 효과적으로 처리하기 위한 워크플로우 방법을 사용하고 있다. ASCI 그리드의 GALE은 모든 자원에 대한 동일한 인터페이스를 목적으로 가지며 제출하는 작업명세에 대해 유연하지 못한 구조를 가지고 있다. 그리고 Condor의 DAGMan은 단순히 작업들간의 입출력 순서만을 기술할 수 있으며 독자적인 기반 시스템을 가지고 있다.

GSFL(Grid Service Flow Language)[8]은 그리드에 웹 서비스의 개념을 추가한 OGSA(Open Grid Service Architecture)[9] 기반에서 웹 기반의 서비스 기술언어를 이용하여 그리드에 대한 웹 서비스와 같은 방법을 적용시켜 그리드를 사용할 수 있는 방법을 제공하는 워크플로우 기술 언어이다. 현재 Globus 3.0에서의 적용을 위해 기술언어를 중심으로 제안되었으며 아직 이를 실행하기 위한 시스템은 아직 구현되지 않고 있다. JSDL (Job Submission Description Language)[10]은 GGF (Global Grid Forum)[11]의 워킹그룹에서 제안한 작업 제출에 관한 연구이다. JSDL은 현재 사용중이거나 또는 개발될 시스템들 사이의 상호작용 기능을 제공하고, 배치 작업과 그 작업에서 요구하는 이질적인 배치 시스템을 수용하는 그리드 환경에 작업을 제출하기 위한 실행 환경을 기술하기 위한 표준화된 작업 제출 언어로서, 표준화 작업을 진행중에 있으며 향후 워킹그룹을 통한 그리드의 배치 작업에 대한 표준화된 기술 방법으로 발전할 것으로 예상된다.

본 논문에서는 Globus가 제공하는 기본 서비스를 바탕으로 일반적인 어플리케이션에서 활용될 수 있는 워크플로우 서비스를 지원하기 위한 메타 스케줄링 프레임워크를 제안한다. MSF는 Globus를 기반 플랫폼으로 이용하며 기존의 어플리케이션을 수정하지 않고 복잡한 작업을 처리할 수 있는 방법을 제공한다. 특히 MSF는 그리드를 필요로 하는 어플리케이션 중에서도 여러 곳

에 분산된 데이타를 이동하여 처리하거나 다단계의 복잡한 처리 절차를 갖는 작업을 배치 작업처럼 흐름을 정의하여 작업을 단계별로 수행하는 어플리케이션에 더욱 효과적인 방법을 제공한다.

3. 메타 스케줄링 프레임워크

MSF의 목적은 Globus 미들웨어의 기능을 확장하여 상위 어플리케이션에서 요구하는 복잡한 작업 환경을 지원할 수 있는 프레임워크를 개발하는 것이다. 이를 위해서 MSF는 사용자 작업의 다양한 요구를 명세할 수 있는 작업 기술 언어가 필요하다. 또한 작업이 사용자가 원하는 대로 처리되고 수행되기 위해서는 해당 작업을 정확하게 기술할 필요가 있다. Globus 2.0버전에서 사용되는 RSL은 간단한 표현 방법만을 제공하여 복잡한 처리 절차를 갖는 작업을 기술하는데 부족한 점이 있다. 그리드 작업이 수행할 내용을 상세하게 표현하여야 그에 대한 정확한 수행을 보장할 수 있다.

그리고 작업의 흐름을 제어하는 워크플로우 관리 시스템이 필요하다, 그리드가 가지는 다양하고 동적인 자원들에 대한 작업의 분배는 매번 달라질 수 있으며, 여러 개의 태스크들로 구성된 작업을 수행해야 할 경우에 이들 태스크간의 작업 순서 또는 데이타의 복사 등과 같은 작업에 필요한 흐름을 적절하게 제어할 수 있어야 한다. 그리드에서 수행될 수 있는 다양한 작업들 중에는 여러 개의 태스크를 단계적인 절차에 의해서 수행해야 하는 경우에 각 태스크의 순서나 데이타 입출력과 관련된 사항을 맞추어서 전체적인 작업의 흐름을 관리할 수 있어야 한다.

마지막으로 Globus와 호환성을 제공하는 하부 구조 등을 가져야 한다. 그리드는 여러 자원들을 표준화된 기술과 프로토콜 등을 이용하여 상위 어플리케이션에 단일한 접근 방법을 제공한다. 이러한 특징은 더 많은 자원을 필요로 하는 연구기관에서 그리드를 필요로 하는 중요한 문제로 부각되었고, 거의 모든 그리드 프로젝트에는 Globus를 이용하고 있다. 따라서 그리드를 지원하는 MSF는 Globus와의 상호작용을 바탕으로 하는 호환성을 지원해야 한다.

MSF의 구조는 크게 4개의 기능별로 구분되며 전체적인 구조는 그림 1과 같다. 사용자는 전용 콘솔을 이용하여 작업에 필요한 정보를 기술한다. 엑세스 관리자(AM)는 사용자 콘솔과 통신하면서 작업 수행에 필요한 사용자 인증, 환경 설정, 작업 제출 대행 등의 서비스를 제공한다. 자원 관리자(RM)는 작업에 필요한 자원을 검색하고 할당하는 서비스를 제공하며 실행 관리자(EM)는 설정된 자원(컴퓨팅 노드)에 대한 작업(실행 파일)을 수행시키고 모니터링한다.

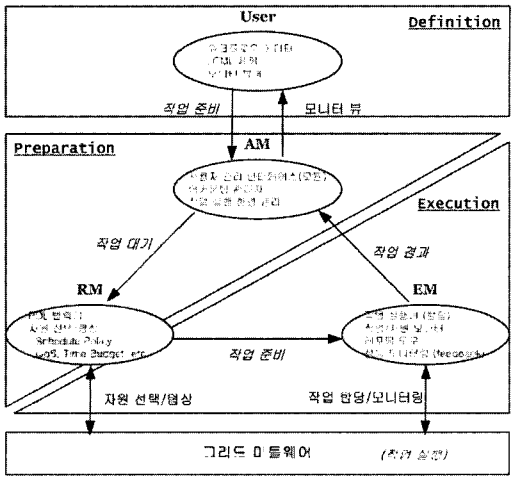

그림 1 메타 스케줄링 프레임워크 구조

MSF는 사용자의 그리드 작업을 실행하기 위해 세 단계(정의, 준비, 실행)의 서비스로 구분한다. 정의 단계는 사용자가 그리드에서 수행할 작업을 정의하는 단계이다. 사용자는 편집기를 이용하여 그리드 작업에 대한 작업 정의 리스트(Job Definition List)를 작성한다. 작업 정의 리스트는 작업 제어 언어(Job Control Markup Language)로 기술된다. 사용자는 준비 단계인 AM에 접속하기 위해서 인증 단계를 거치며, 인증에 성공하면 프록시를 생성하여 그리드에 대한 작업의 인증 절차를 마무리한다.

두 번째 준비 단계는 그리드 작업을 실행시키기 위해서 자원의 할당을 준비하는 단계이다. AM은 인증된 사용자로부터 받은 작업 정의 리스트를 검사하여 사용자 작업에 대한 에이전트를 생성한다. 사용자 에이전트는 사용자가 제출한 작업에 대한 실행 과정을 추적하며 사용자의 요구에 대한 서비스를 대신 수행하는 역할을 한다. RM은 작업 정의 리스트를 분석하여 작업에 필요한 그리드 자원을 선택한다. 자원 선택 방법은 외부 알고리즘을 사용자가 플러그인 형태로 선택할 수 있으며 MDS[12], NWS[13] 등 외부의 그리드 서비스를 이용한다.

마지막으로 실행 단계는 선택된 자원에 대한 작업의 할당과 순차적인 작업 처리를 수행하는 단계이다. 준비 단계에서 처리된 정보는 작업 리스트를 통하여 실행 단계로 이전된다. 작업 리스트에는 그리드 작업의 처리 절차와 입·출력 데이타, 할당될 자원의 정보를 포함하고 있다. EM은 순서를 갖는 작업 정보에 따라 해당 그리드 자원에 작업을 스케줄링하며 입·출력 데이타를 관리한다. 그리고 작업의 상태 모니터링과 리포팅 기능을 제공한다.

3.1 작업 제어 언어와 편집 도구

그리드 작업을 기술하기 위해 작업 정의 도구는 사용자가 편리하게 작업을 표현할 수 있는 방법을 제공해야 한다. 그리고 다양한 그리드 환경에 대해 조건을 적용할 수 있는 유연한 기술 방법이 필요하며, 태스크와 입출력 데이타 등의 의존 관계에 대한 표현 방법을 제공해야 한다.

JCML은 다양한 시스템에서 자료의 호환성을 유지하기 위해 XML 기반의 문서 형식으로 구성한다. XML은 자료의 호환성을 유지할 수 있으며, 또한 요소(Element) 내에 속성과 다른 요소를 포함할 수 있는 구조적인 특징을 가지고 있다. JCML은 각 구성 요소들의 의존 관계를 기술하기 위해 그래프 방식으로 XML을 표기할 수 있는 GXL(Graph eXchange Language)[14]을 이용하였다. GXL은 소프트웨어에서 표현되는 그래프 방식의 정보를 다른 형식으로 변경할 수 있는 형식을 제공하기 위해 개발된 XML 기반의 언어이며, 노드(Node)와 에지(Edge)로 표현되는 그래프를 XML 스키마를 이용하여 다양한 형식으로 변형시킬 수 있다. JCML은 사용자에게 내부적인 복잡함을 감추고 작업을 기술할 수 있는 간단한 조작 방법을 제공하기 위해 그래픽 인터페이스를 같이 제공한다. JCML은 GXL 기반의 그래프 방식으로 설계되었기 때문에 그래픽 화면에서 보여주는 그래프를 이용한 노드와 에지 아이콘을 사용하여 사용자의 이해를 높일 수 있다. JCML은 XML 스키마로 정의되며 그림 2와 같은 구조를 갖는다.

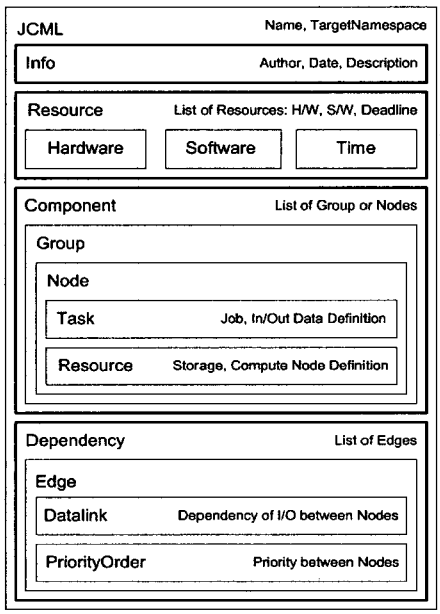

그림 2 JCML 구조

문서 정보 (Info)

JCML 문서의 이름과 문서의 범위, 요소에서 사용되는 네임스페이스를 갖는다. 그리고 작성한 사용자와 작성일, 간단한 설명을 포함한다.

자원 정보 (Resource)

Resource 요소는 JCML에서 기술되는 작업이 수행되기 위해 필요로 하는 하드웨어와 소프트웨어, 시간 요소로 구성된다. 하드웨어 요소는 컴퓨터 구조, 중앙 처리 장치, 메모리, 디스크 용량, 네트워크의 대역폭 요소로 구성되고, 소프트웨어 요소는 운영체제, 설치된 어플리케이션, 작업 스케줄러 요소로 구성된다. 시간 요소는 사용자 작업이 종료되기를 희망하는 시간을 나타낸다.

작업 구성 (Component)

Component 요소는 응용 프로그램이 정상적으로 수행되기 위한 여러 단계의 프로세스들을 표현하는 단위이며 노드(Node)와 그룹(Group)으로 구성된다. 노드는 태스크 노드와 자원 노드로 구분된다. 태스크 노드는 실제 작업이 실행되는 주체를 나타내며 일반적으로 컴퓨터 시스템에서 수행되는 단일 프로세스라고 할 수 있다. 자원 노드는 태스크 노드가 수행되기 위해 필요한 보조 노드로서 프로세스가 실행되기 위해 필요한 자원(저장 장치, 컴퓨터 등의 물리적 자원 장치)을 나타낸다. 태스크 노드는 실행 파일, 입출력 데이타, 환경 변수를 포함한다. 자원 노드는 해당 자원에 대한 태스크 노드가 접근하여 사용할 수 있는 정보를 포함하고 있다. 그룹은 그리드 작업을 프로세스에 따라 분리할 필요가 있을 경우 한개 또는 여러 개의 노드를 묶어 지정할 수 있는 논리적인 노드이다. 지정된 그룹은 외부에서는 하나의 노드로 인식되어 다른 노드와의 상호 작용을 수행한다. 그리고 그룹은 다른 그룹을 포함할 수 있다.

그림 3은 JCML을 이용하여 작업 예제를 모델링한 것이다. 그림에서 실선의 원형은 노드를 표시하고 있다. 전체 작업은 저장 장치 S1, S2와 태스크 A, B, 그룹 A로 구성된다. 그룹 A는 다시 4개의 태스크 노드 C, D, E, F로 구성된다. 그룹 A에 속해있는 태스크들은 사용자가 다른 작업 프로세스와 별도의 구분이 필요하다고 결정하는 경우, 다른 노드와 분리되어 그룹으로 지정될 수 있다. 분리된 그룹은 다른 노드의 수행과는 상관없이 별도의 수행 과정을 거칠 수 있고, 또한 다른 노드와의 연관 관계를 가질 수 있다. 그림 3의 예제에서는 태스크 B에 의해 그룹 A의 노드들이 작업을 실행하도록 표기되고 있다. 노드들 간의 의존 관계에 대한 내용은 다음 소절에서 설명한다.

의존 관계 (Dependency)

Dependency 요소는 component 요소를 구성하는 그룹 또는 노드들 간의 관계를 기술하는 에지(Edge)로 구

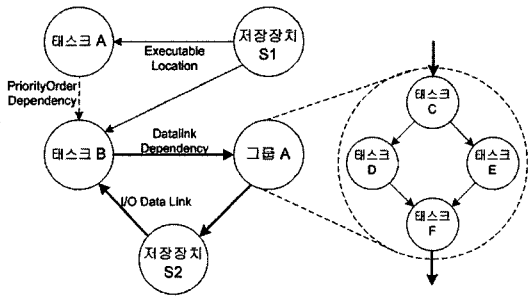

그림 3 JCML 모델 예제

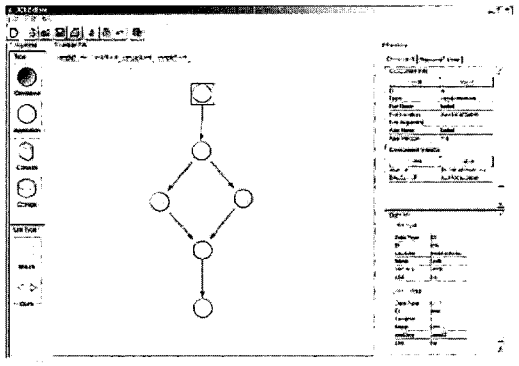

그림 4 JCML 편집기

성된다. 에지는 그룹 또는 노드를 연결하여 두 노드(또는 그룹)의 실행 순서, 입출력 데이타의 의존 관계 등을 표시한다.

에지는 우선 순위(PriorityOrder)와 데이타 링크(Datalink)로 구성된다. 그리고 에지에는 방향(direction) 속성을 가지고 있어 시작점(from)과 끝점(to)으로 연결된 두 노드의 순서를 나타낸다. 우선 순위는 에지의 방향 속성에서 표시되는 두 노드나 그룹에 대한 실행 우선 순위만을 표시한다. 데이타 링크는 우선 순위가 갖는 속성외에 두 노드간에서의 데이타 이동을 나타낸다. 데이타 링크는 태스크 노드에 포함된 입출력 데이타 요소를 참조하여 태스크 노드가 실행되기 위한 데이타 관계를 표시하여 워크플로우 관리 시스템이 이를 처리할 수 있도록 알려준다.

그림 3에서 각 노드간의 연결된 화살표는 에지를 나타내고 있다. 모든 에지는 방향을 가지고 있는 화살표로 이루어져 있으며 시작점과 끝점을 나타내고 있다. 점선으로 된 에지는 노드간의 우선 순위를 표시한다. 우선 순위 에지는 태스크 또는 그룹 노드 간에만 사용되며 자원 노드에서는 사용되지 않는다. 태스크 A와 B는 저장 장치 S1과 실선 에지로 연결되어 있다. 이 에지는 각 태스크를 수행하기 위한 실행 파일이 있는 위치를 지정하는 것이며, 실제 태스크가 실행될 때에는 자원 스케줄러에서 선택된 컴퓨팅 자원으로 이동된다. 태스크 B와 그룹 A의 실선 에지는 B의 출력 데이타가 그룹 A의 입력 데이타로 사용되는 입출력 데이타의 의존 관계를 데이타 링크 요소로 표시한다. 그리고 저장 장치 S2와 연결된 에지는 태스크 B의 입력으로 S2의 데이타를 이용하며 그룹 A의 출력 데이타를 S2에 저장한다는 것을 표시하고 있다. 그룹 A의 내부 태스크 C, D, E, F는 외부(태스크 노드 B)로부터의 입력을 C가 받아 수행한 다음 출력 데이타를 태스크 D, E의 입력으로 받아 각 출력을 F의 입력으로 넘겨준다. 태스크 F의 출력은 다시 저장 장치 S2에 저장된다.

편집기는 사용자가 직관적으로 이해할 수 있는 구성을 가져야 하며 간단한 동작만으로도 작업정의 리스트를 작성할 수 있는 편리함이 같이 제공되어야 한다. JCML 편집기는 사용자 작업을 하나의 객체로 나타내는 아이콘으로 표현하여, 마우스로 필요한 아이콘을 선택하여 작업 화면에 드래그 앤 드롭만으로 간단한 작업정의 리스트를 작성할 수 있다. 그림 4는 작업 정의 리스트를 작성중인 편집기 화면이다.

편집기는 크게 3부분으로 구성되어 있다. 좌측의 도구 모음 창에는 작업 정의 리스트에서 사용되는 노드와 에지의 종류를 표시한다. 아이콘 모양의 노드는 작업의 실행 형태와 컴퓨팅 자원의 종류로 분류하며, 화살표 형태의 에지는 단방향과 양방향의 데이타 이동과 같은 노드 사이의 관계를 표시한다. 가운데의 작업 창은 사용자가 원하는 형태의 작업 형태를 표현한다. 좌측의 아이콘 모음을 이용하여 작업에 필요한 프로그램들과 외부 저장장치를 선택하여 배열한 다음, 에지를 이용하여 데이타의 이동과 같은 의존 관계를 표시한다. 우측의 속성 창은 작업 창에서 선택된 객체의 속성을 기입할 수 있는 부분으로 실행 프로그램 이름, 위치, 실행 옵션, 입출력 데이타 위치 등과 같은 내용을 기술할 수 있다.

사용자는 좌측의 아이콘을 마우스를 이용한 드래그 앤 드롭으로 작업 창에 끌어다 놓고, 화살표를 이용하여 미리 선택한 두개의 노드 객체 사이에 에지를 연결한다. 각 객체에 속성 값을 입력하기 위해서는 먼저 마우스로 객체를 선택한 후 우측의 속성 입력창을 이용하여 입력한다.

3.2 워크플로우 관리 시스템

워크플로우 관리 시스템은 작업 프로세스에 대한 실행과 관리를 위한 운영 환경을 제공한다. MSF는 프로세스에 대한 상태 전이(state transition)에 기반한 스케줄러 기반 패러다임(Scheduler-based paradigm)의 워크플로우 서비스를 제공한다. 스케줄러 기반 패러다임은 작업 프로세스를 활성화하여 태스크로 변환될 수 있는

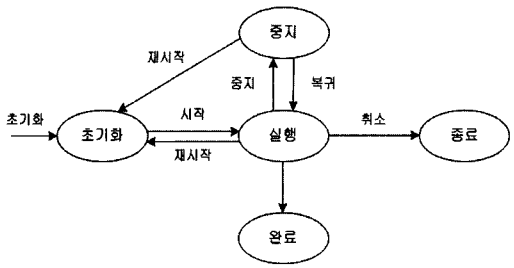

그림 5 워크플로우 태스크의 상태 전이도

가장 작은 실행 단위인 액티비티(activity)들로 나누어서 프로세스 처리기에 할당하여 실행시키는 방법이다[15].

태스크는 초기화, 실행, 중지, 완료 또는 종료 상태를 갖는다. 태스크는 자원에 할당되어 실행되기 위해 초기화되고 실제 자원에 할당되어 실행된다. 실행 중인 태스크는 외부 상황에 따라 중지되거나 종료된다. 한번 종료되면 다시 재실행되지 못하며 중지된 태스크는 복귀되어 실행된다. 실행 또는 중지중인 태스크는 에러와 같은 미리 예상되지 않은 상황에 의하여 재실행될 수 있다. 실행 환경으로 인한 에러가 발생하여 태스크가 정상적으로 수행될 수 없는 상황이 발생하면 재실행되기 위해 초기화 단계로 되돌아간다.

작업이 자원을 할당받아서 실제로 로컬 관리자에 의해 스케줄링되기 위해서는 작업의 분석과 자원 검색 및 선택, 작업 리스트의 생성, 실행 엔진에 의한 해석과 실행 단계를 거친다. 워크플로우 관리 시스템에서 사용자 작업을 처리하기 위해 수행되는 과정은 그림 6과 같다.

사용자는 편집 시스템을 이용하여 수행될 작업에 대한 내용을 포함하는 작업 정의 리스트를 작성한다. 작업 정의 리스트는 JCML 방법에 따라 수행될 작업의 내용

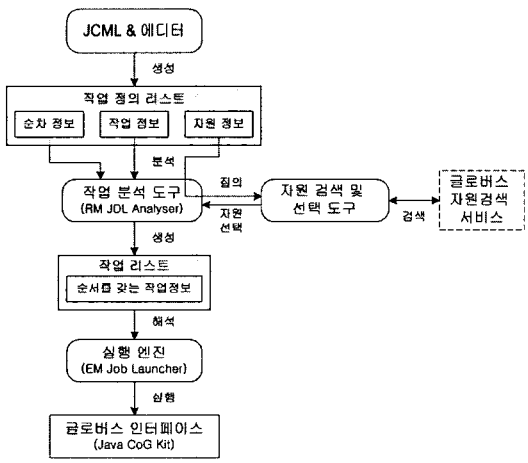

그림 6 워크플로우 서비스

(실행 파일명, 위치, 명령 인자, 입출력 데이타, 소프트웨어)과 다른 태스크들과의 연관 관계(실행 순서, 입출력 데이타의 의존관계), 자원의 명세를 포함한다.

작업 분석 도구에서는 작업 정의 리스트를 분석한다. 작업 정의 리스트에 기술된 자원 정보에 대한 내용을 분석하기 위해 자원 검색 도구를 이용하여 Globus의 디렉토리 서비스를 받아 자원을 검색한다. 선택된 자원과 해당 자원에서 수행할 작업 내용, 작업 순서를 결합하여 작업 리스트를 생성한다. 작업 리스트에는 작업을 구성하는 여러개의 태스크가 실행될 자원들에 대한 정보와 함께 결합되어 있으며 각각의 태스크에 대한 실행 순서 정보가 같이 포함되어 있다. 그리고 각 태스크는 순서를 갖는 최소한의 실행 단위인 액티비티들로 구성된다. 작업 리스트는 XML 구조를 이용하여 설계되었으며 그 구조는 그림 7과 같다.

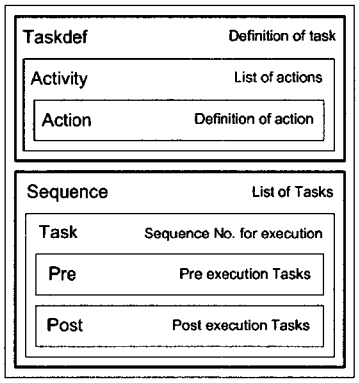

그림 7 작업 리스트 구조

Taskdef는 하나의 작업을 구성하는 서브 태스크의 정의이며 여러개의 순서를 갖는 Action들로 구성된 Activity를 갖는다. Action은 실행 프로그램 이외에 태스크가 수행하기 위한 전처리와 후처리 과정을 나타낸다. 예를 들면 실행 프로그램에서 필요로 하는 입력 데이타나 결과값인 출력 데이타를 다른 곳으로 복사/이동하는 명령이다. Sequence는 정의된 태스크의 순서 정보를 갖는다. 일련번호를 갖는 태스크는 해당 태스크를 기준으로 이전에 수행되어야 하는 태스크와 이후에 수행되어야 할 태스크를 표시함으로써 태스크들 간의 의존관계를 표현할 수 있다.

실행 엔진은 작업 리스트에 정의된 순서대로 액티비티들을 수행한다. 실제 작업을 로컬 스케줄러에서 수행하기 위해서 실행 엔진은 XML로 기술되어 있는 작업 리스트의 실행 정보를 Globus의 RSL 형태로 바꾸어서 수행한다. 그림 8은 작업 정의 리스트(a)를 작업 리스트(b)로 가공한 뒤에 Globus RSL(c)로 전환되는 과정을

```
<node id="d" type="applicationJob">
  <excutable name="mkdpf3" location="/usr/local/autodock/share/" arguments=""/>
  <application name="autodock" version="3.0.1" />
  <configuration>
    <env name="workDir" value="/home/seventy9/queue/job"/>
    <envname="PATH"  value="/bin:/user/bin:/bin/usr/local/autodock/bin:/usr/local/
                                                              autodock/share"/>
  </configuration>
  <data>
    <in id="dia" name="l.pdbq" location="" cmdArg="" link="yes"/>
    <in id="dib" name="p.pdbqs" location="" cmdArg="" link="yes"/>
  </data>
</node>

<edge id="bd" from="b" to="d" direction="directed" type="datalink">
</edge>
<edge id="de" from="d" to="e" direction="directed" type="datalink">
</edge>
```

(a) 작업 정의 리스트

```
<action id="3">
  <execution>
    <resource>
      <host name="203.253.23.52" port="2119"/>
      <workDir path="/home/seventy9/queue/job"/>
      <env name="PATH" value="/bin:/usr/bin:
          /bin/usr/local/autodock/bin:/usr/local/autodock/share"/>
    </resource>
    <command>
      /usr/local/autodock/share/mkdpf3 l.pdbq p.pdbqs
    </command>
  </execution>
</action>
```

(b) 작업 리스트

```
&(directory="/home/seventy9/queue/job")
 (arguments="l.pdbq""p.pdbqs")
 (executable = "/usr/local/autodock/share/mkdpf3")
 (environment=("PATH" "/bin:/user/bin:/bin/usr/local/autodock/bin:
  /usr/local/autodock/share"))
```

(c) Gloubs RSL

그림 8 MSF의 작업 정의 리스트와 작업 리스트, RSL

보인 예이다.

3.3 MSF의 구성

MSF는 작업의 구성, 자원의 할당, 작업의 실행 서비스 등을 통합하여 사용자에게 제공하는 상위레벨의 미들웨어를 구성하는 시스템이다. 3장에서 언급하였듯이 MSF는 3단계로 구분된 서비스를 제공하며 다른 시스템과의 확장을 고려하여 각 서비스와 기능별로 독립된 기능을 갖는 구성요소를 갖는다. 그림 9는 MSF 구성요소의 기능별 배치를 보인다. 사용자와 시스템 사이의 연결을 관리하며 작업의 정의부분을 담당하는 엑세스 관리자, 정의된 작업을 분석하고 자원을 할당/관리하는 준비 부분을 담당하는 자원 관리자, 준비된 작업을 실제 로컬 자원에 스케줄링하는 수행 부분을 담당하는 실행 관리자, 마지막으로 MSF 시스템내의 모든 정보를 통합하여 관리하는 정보 관리자로 구성된다.

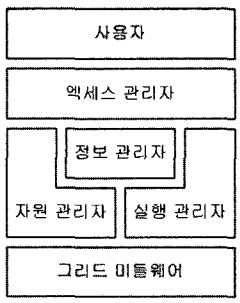

그림 9 MSF 구성요소

엑세스 관리자(AM)는 그리드 사용자와의 인터페이스를 제공하는 MSF의 구성요소 중 하나이다. AM의 주요기능은 사용자의 접속을 관리하며 사용자가 수행하려는 작업에 대한 제출, 제어 관리, 모니터링 등을 대행하고 그림 10과 같은 구조를 갖는다. 사용자가 AM에 접속하면 인증 라이브러리에 의해 인증을 받는다. 인증이 정상적으로 수행되면 사용자 에이전트 쓰레드를 생성하여 접속된 사용자와 일대일로 대응하며 모든 사용자 관련 서비스를 수행한다. 사용자 에이전트는 제출된 작업을 다음 단계인 RM에게 넘기고 작업의 단계별 절차와 작업 노드에서의 수행 상태를 모니터링하고 최종적으로 결과 파일을 모아서 사용자에게 보여준다. AM 내에서

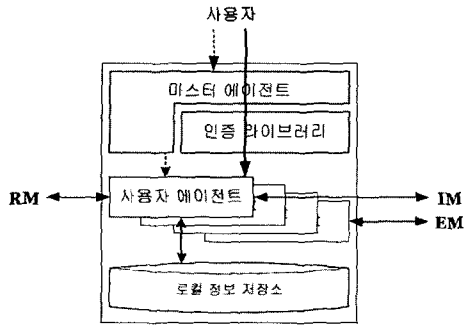

그림 10 엑세스 관리자 구조

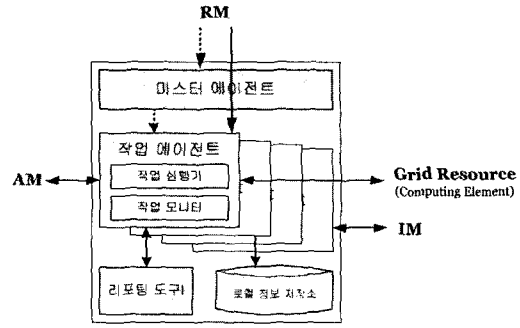

그림 12 실행 관리자 구조

발생한 모든 이벤트는 로컬 정보 저장소에 저장되며 작업 수행과 관련한 이벤트는 정보 관리자(IM) 인터페이스를 통하여 외부에 저장한다.

자원 관리자(RM)는 AM으로부터 받은 작업 정의 리스트를 분석하여 외부 서비스를 이용하여 요구되는 자원을 선별한다. 작업 정의 리스트는 JDL 번역기에 의해 분석되고 자원 정보 수집기는 분석된 자료를 바탕으로 작업 파일에서 요구되는 자원의 정보를 각 정보원으로부터 수집한다. 자원 선택기는 수집된 자원의 정보와 작업 파일의 요구사항을 결합하여 적당한 자원에 대한 실제 가용여부와 예약(옵션)을 수행하는 협상 모듈을 실행하여 최적의 자원을 찾아낸다. 분석된 작업 파일과 자원 리스트를 EM에게 보낸다. 그림 11은 자원 관리자의 구조를 나타낸다.

실행 관리자(EM)은 RM에서 선택된 자원들에게 사용자 작업을 실제로 할당하여 실행시키고 작업의 진행 상태를 모니터링한다. 그림 12에서와 같이 RM에서 작업 수행 요청을 받은 마스터는 작업 당 하나의 작업 에이전트를 생성하여 사용자 작업과 일대일로 대응한다. RM에서 선택된 작업 노드에 연결을 시도한 후 작업을 할당하며 작업 중에 발생할 수 있는 모든 상황을 모니

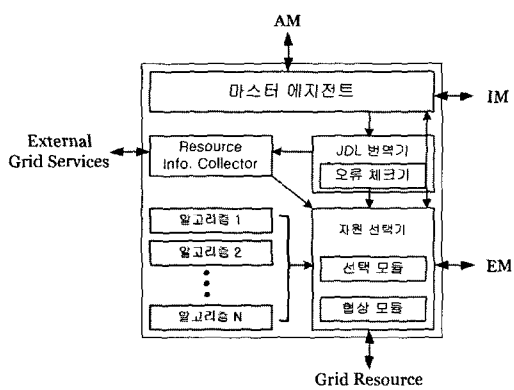

그림 11 자원 관리자 구조

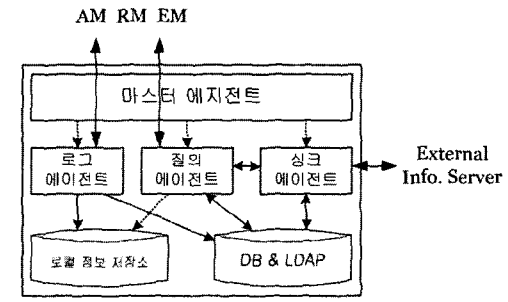

그림 13 정보 관리자 구조

터링한다. 모든 이벤트 데이타는 로컬 정보 저장소에 저장하고 작업수행과 관련된 이벤트는 IM 인터페이스를 통하여 외부에 기록한다.

정보 관리자(IM)는 MSF의 다양한 구성요소(AM, RM, EM)에서 서비스 수행 시 발생하는 각종 이벤트와 로그를 기록하고 관리하는 서비스를 제공한다. 정보 관리자의 구조는 그림 13과 같다. 로그 에이전트는 외부에서 발생하는 이벤트를 데이타베이스에 저장하고 질의 에이전트는 외부에서 요청하는 정보에 대한 검색 기능을 제공한다. 싱크 에이전트는 정보 관리자가 복수로 지정되어 운영되거나 사용자 인증과 같은 외부 환경에 대한 변화를 MSF가 관리하는 데이타에 반영하기 위한 기능을 제공한다. 정보 관리자는 여러 사용자가 동시에 여러 작업을 수행시킬 때 발생할 수 있는 문제와 각종 서비스 구성 요소들에서 발생할 수 있는 문제를 해결하기 위한 기본 정보를 제공한다. 단일화된 IM 인터페이스를 통하여 각종 서비스에서 발생하는 이벤트를 수집하여 데이타베이스에 저장하고 필요한 정보를 검색한다.

4. MSF의 구현

MSF는 자바 언어를 이용하여 구현하였으며, Globus와의 호환성을 위해 Java CoG Kit을 이용하여 Globus 인터페이스를 구현하였다. 그리고 JCML과 작업 리스트

의 파싱을 위한 도구로는 Xerces를 이용하였다.

테스트를 위해서는 생명 공학 분야에서 신약을 설계할 때 사용되는 가상 검색 어플리케이션(AutoDock [16])을 이용하였다. 가상 검색은 특정 질병에 관여하는 단백질이 활성화된 곳에 잘 반응할 것이라고 예상되는 화합물을 하나씩 결합하는 도킹(docking)이라는 과정을 통하여 신약 후보 물질을 검색하는 방법이다. 가상 검색을 하기 위해서는 화합물이 가지고 있는 고유한 형태를 도킹에 필요한 형태로 바꾸는 과정을 여러 단계 수행해야 한다.

AutodDock의 실행과정과 이를 JCML에 적용하기 위한 모델링은 각각 그림 14, 15와 같다. 모델링후에 JCML 편집기를 이용하여 앞절의 그림 4와 같이 작업 내용을 표현한다. 작성된 JCML과 워크플로우 관리 시스템에서 생성되는 각 태스크에 대한 자원의 할당과 실행 순서 정보를 갖는 작업 리스트, 실제 로컬 자원에서 스케줄링하기 위해 Globus의 RSL로 변환되는 과정을 그림 8에 나타내었다.

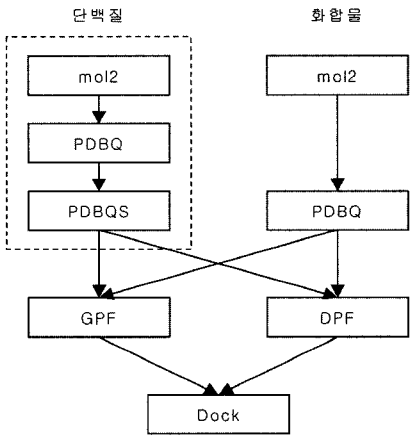

그림 14 Autodock 실행 과정

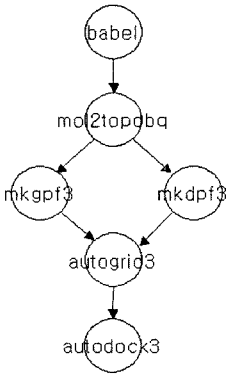

그림 15 프로세스 모델

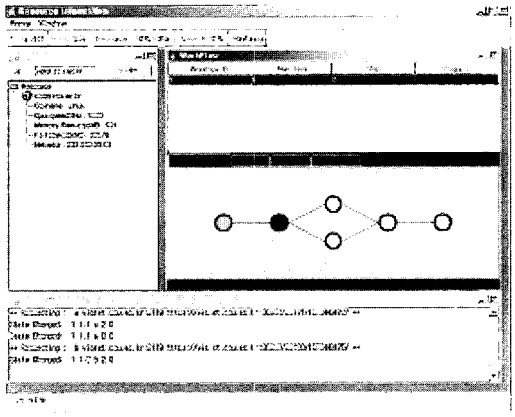

그림 16 MSF 메인 콘솔

그림 16은 AutoDock을 MSF에서 실행하는 과정을 보이는 메인 콘솔 화면이다. 좌측의 트리구조 메뉴는 현재 실행중인 태스크가 수행 또는 예정인 자원들의 정보가 출력된다. 우측은 전체 작업에 대한 진행 정보와 함께 중간의 그래프는 사용자가 JCML로 작성한 작업 프로세스의 과정을 표시하며 각 태스크(원형의 모형)의 진행 상태를 서로 다른 색으로 표시하며 보다 자세한 메시지가 하단부에 로그로 기록된다.

5. 결론 및 향후 연구 내용

MSF는 다양한 그리드 작업의 처리 서비스를 제공하기 위해 워크플로우 기반의 그리드 관리 시스템을 제공한다. MSF가 갖는 특징은 다음과 같다. 기본적으로 그래픽 기반의 인터페이스를 제공하며, 직관적인 해석과 쉽게 이해될 수 있는 그래프 방식의 표현법을 제공하여 작업에 대한 내용과 처리 절차 등을 자세하게 표현할 수 있는 사용자 작업 환경을 제공한다. 그리고 복잡한 그리드 작업을 처리하는 워크플로우 관리 서비스의 제공으로 다양한 요구를 갖는 기존의 어플리케이션을 수정없이 사용할 수 있으며 단계별 작업에 대한 제어 방법 지원을 통해 사용되는 자원의 낭비를 줄일 수 있다. 마지막으로 그리드 아키텍처에서 MSF 시스템은 미들웨어 서비스를 이용한 상위 서비스 요구에 대한 서비스를 제공할 수 있는 기반 프레임워크를 제공하여, 새롭게 추가되는 서비스에 대한 연계를 통한 서비스도 수용할 수 있는 확장성을 제공한다.

본 논문에서 제안하는 MSF는 기존 Globus 미들웨어를 그대로 이용하여 어플리케이션에서 요구하는 다양한 서비스에 대해 기본적인 확장성을 갖는 프레임워크를 제공하는 것을 목표로 하고 있다. 앞으로 MSF에서 제공하는 프레임워크를 이용하여 어플리케이션 또는 사용자 중심으로 좀더 세분화된 기능을 제공할 수 있는 연

구가 필요하다. 현재 글로벌 그리드 포럼은 세분화된 워킹그룹 단위로 그리드의 표준화와 기능 확장을 위해 계속적인 연구가 진행되고 있다. MSF 시스템도 GGF의 연구를 바탕으로 호환성과 확장성을 제공하기 위한 연구를 지속적으로 수행해야 한다. 그리고 OGSA 기반의 그리드 웹 서비스를 지원하기 위한 웹 표준을 위한 시스템의 기능 확장과 새로운 어플리케이션에서 요구되는 서비스에 대응하여 다양한 기능을 제공할 수 있는 서비스의 확장에 대한 연구를 수행하여야 할 것이다.

참 고 문 헌

[1] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

[2] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal supercomputer Applications, Vol. 11, No. 2, 1997.

[3] L. Fisher (eds.), Workflow Handbook, Workflow Management Coalition, 2002.

[4] J. Cao, S. A. Jarvis, S. Saini, G. R. Nudd, "GirdFlow: Workflow Management for Grid Computing," Proceeding of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 12-15, 2003.

[5] R. Stevens, A. Robinson and C. Goble, "myGrid: personalised bioinformatics on the information grid, Bioinformatics," Vol. 19, Suppl. 1, pp. 302-304, 2003.

[6] J. Geiriget, H. Bivens, S. Humphreys, W. Johnson, R. Rhea, "Constructing the ASCI Computational Grid," http://vir.sandia.org/drmweb/docs/grid-hpdc00.pdf.

[7] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Cluster Computing, Vol. 5, No. 3, pp. 237-246, 2002.

[8] S. Krishnan, P. Wagstrom, G. Laszewski, "GSFL: A Workflow Framework for Grid Services," http://www.globus.org/cog/papers/gsfl-paper.pdf.

[9] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum, June 2002.

[10] JSDL, JOb Submission Description Language Working Group, Global Grid Forum, 2003.

[11] GGF, Global Grid Forum, http://www.ggf.org/.

[12] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing," Proceeding of the 10th IEEE International Symposium on High-Performance Distributed Computing, 2001.

[13] R. Wolski, N. spring, J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757-768, 1999.

[14] Richard C. Holt, Andreas Winter, "A Short Introduction to the GXL Software Exechange Format," Proceeding of the 7th Working Conference on Reverse Engineering, 2000.

[15] A. Cichocki, A. S. Helal, M. Rusinkiewicz, D. Woelk, Workflow and Process Automation: Concepts and Technology, Kluwer, 1998.

[16] AutoDock, http://www.scripps.edu/pub/olson-web/autodock/.

황 석 찬

2003년 숭실대학교 컴퓨터학과(박사). 2003년~2004년 숭실대학교 시스템소프트웨어연구실 박사후과정. 2004년 한국과학기술정보연구원 선임연구원. 관심분야는 분산/병렬 컴퓨팅, 그리드 컴퓨팅, 시스템 소프트웨어, 미들웨어

최 재 영

정보과학회논문지 : 컴퓨팅의 실제 제 10 권 제 2 호 참조