ISBN 978-89-6211-675-5

계산화학 교육 실습 및 연구를 위한 슈퍼컴퓨팅 기반 통합 플랫폼 구축

한 영 만

hans@kisti.re.kr

한국과학기술정보연구원

Korea Institute of Science and Technology Information

KiSTi Korea Institute of Science and Technology Information
www.kisti.re.kr

목 차

표 차례

그림 차례

1. 서론

계산화학은 분자동역학, 양자화학과 같은 이론화학에서의 수학적 방법론을 기초로 한 컴퓨터 프로그램을 사용하여 분자 구조 및 특성 등에 대한 실험적 사실을 증명하거나 예측하는 학문이다. 계산화학은 실험을 통해 확인하기 어려운, 또는 높은 비용을 요구로 하는 실험을 전산 모사하여 실험결과를 예측할 수 있어 매우 유용하게 활용되어 질 수 있는 분야이다. 게다가 컴퓨팅 능력이 기하급수적인 발달로 인해 대부분의 화학분야에서 널리 사용되고 있다. 계산화학을 통하여 전자 밀도, 분자 구조 최적화, 진동수, 전이 상태, 단백질 상호작용, 전하분포, 포텐셜 에너지, 열역학적 화학 계산 등을 실험 이전에 미리 계산해 낼 수 있다. 화학적 문제들에 대한 계산 화학의 접근 방법은 크게 *ab initio*와 semi-empirical한 방법으로 구분할 수 있다. *ab initio* 방법은 슈레딩거 방정식을 풀어 분자 구조를 예측하는 것이고 semi-empirical 방법은 실험적인 데이터를 수학적 모델에 적용하는 근사법이다. 여기에는 분자역학(Molecular Mechanics)과 같은 고전 역학을 응용하여 원자와 분자 구조를 근사하는 방법도 포함된다. <표 1>은 계산화학 분야의 접근 방법의 장단점 및 적합한 적용 방법을 요약한 것이다[1].

구분	장점	단점	적용
Molecular Mechanics	- 적은 컴퓨팅 자원으로 빠르게 계산할 수 있음 - 큰 분자 구조 계산에도 사용될 수 있음	- 제한된 분자에 대한 force field 계산만 가능 - 전자상태 계산 불가능 - 실험 데이터가 입력 파라미터로 요구됨	- 수천 개 원자로 구성된 큰 시스템 - 원자간 결합이 끊어지거나 생성되지 않는 시스템
Semi-empirical	- *ab initio* 계산보다 적은 컴퓨팅 자원으로 빠른 계산을 할 수 있음 - 전이 상태나 여기 상태 계산 가능	- 실험 데이터가 파라미터로 요구됨 - *ab initio* 계산보다 덜 정확함	- 수백 개 원자로 구성된 중간 정도 시스템 - 전자 전이가 일어나는 시스템

Ab initio	- 실험 데이터가 필요 없음 - 다양한 계산 영역에서 사용가능 - 전이 상태나 여기 상태 계산 가능	- 높은 컴퓨팅 능력이 요구됨	- 10개미만의 원자로 구성된 작은 시스템 - 실험데이터가 없는 이론적 시스템 - 정확한 결과가 요구되는 시스템

표 1 계산화학 분야의 접근 방법

컴퓨팅 능력의 기하급수적인 발달과 계산화학 기법의 발달에도 불구하고 계산화학 분석 도구가 실습 및 연구 환경에 활용되기에는 몇 가지 문제점이 있다. 첫째, 계산화학 분석 도구 설치 및 유지가 어렵다는 점이다. *ab initio* 계산을 수행하는 대표적 프로그램 중 GAMESS를 제외한 Gaussian, NWChem, Q-Chem 등은 모두 상용 라이선스이며 설치 및 유지보수가 까다로운 편이다. 둘째, 대부분의 계산화학 프로그램 들은 Unix를 기반으로 하기 때문에 IT 비전공자인 연구자들이 Unix 명령어를 습득하여 활용하기에는 매우 부담스러운 일이다. 게다가 복수의 계산화학 도구를 활용하여 연구를 진행할 경우 각 입출력 양식이 각기 다른 도구들을 관리하고 활용하기는 매우 어려운 일이다. 셋째, 여러 단계로 구성된 계산화학 분석인 경우 입출력 데이터 양식이 상이한 프로그램 간의 연계가 어렵다는 것이다. 넷째, 단백질 접힘 등과 같은 비교적 큰 분자에 대한 복잡한 계산은 개인 PC 환경에서는 거의 불가능 하므로 슈퍼컴퓨터와 같은 대규모 컴퓨팅 환경이 필요하다. 그러나 이와 같은 조건에 부합하는 전산자원들은 하드웨어 자체의 단가가 매우 높을 뿐 아니라, 이를 효과적으로 관리하고 활용할 수 있는 전문 인력을 요구하기 때문에 개별적인 연구실 단위로는 제대로 된 연구기반을 갖추는 것이 현실적으로 불가능한 실정이다. 다섯째, 사용자 간 자료 공유 및 협력 연구가 어렵다는 점이다. 계산 화학 실습의 경우 사용자 간 출력 데이터 공유가 필수적이고 이전 계산 과정의 재활용 및 추가 기능 보완 등의 작업이 필요하다. 따라서 사용자의 데이터 및 계산 과정의 통합적 관리와 사용자 간의 데이터 공유 환경이 절실히 요구된다.

본 연구보고서에서는 이러한 문제점들을 극복하고 보다 효과적이고 확장적인 계산화학 실습 및 연구를 위한 슈퍼컴퓨팅 기반 통합 플랫폼 (Chemworks) 구축 방안에 대해 제시하고자 한다.

2. 계산화학 실습/ 연구 시나리오 및 요구기능 도출

2.1. 계산화학 실습 시나리오

<그림 1>은 Chemworks 시스템을 활용한 계산화학 실습 과정에 대한 시나리오를 도식화 한 것이다. <그림 1>에서 보여주는 바와 같이 일반적인 계산화학 실습 과정은 다음의 단계로 이루어진다고 할 수 있다.

- Tutor는 특정 과제와 관련한 분석 파이프라인(워크플로우)과 입력 샘플 자료를 Chemworks 클라이언트 프로그램을 통하여 작성/등록하고 해당 클래스의 Student 들과 공유한다.
- Student는 공유된 Tutor의 워크플로우와 샘플 입력 자료를 Chemworks 클라이언트를 통하여 검색하고 복사 및 편집한다.
- Student는 편집한 워크플로우에 입력 값을 설정하고 실행하여 결과물을 생성/저장하고 Tutor에게 공유(제출)한다.
- Tutor는 Student로부터 제출된 계산 결과물을 복사하고 평가한다.

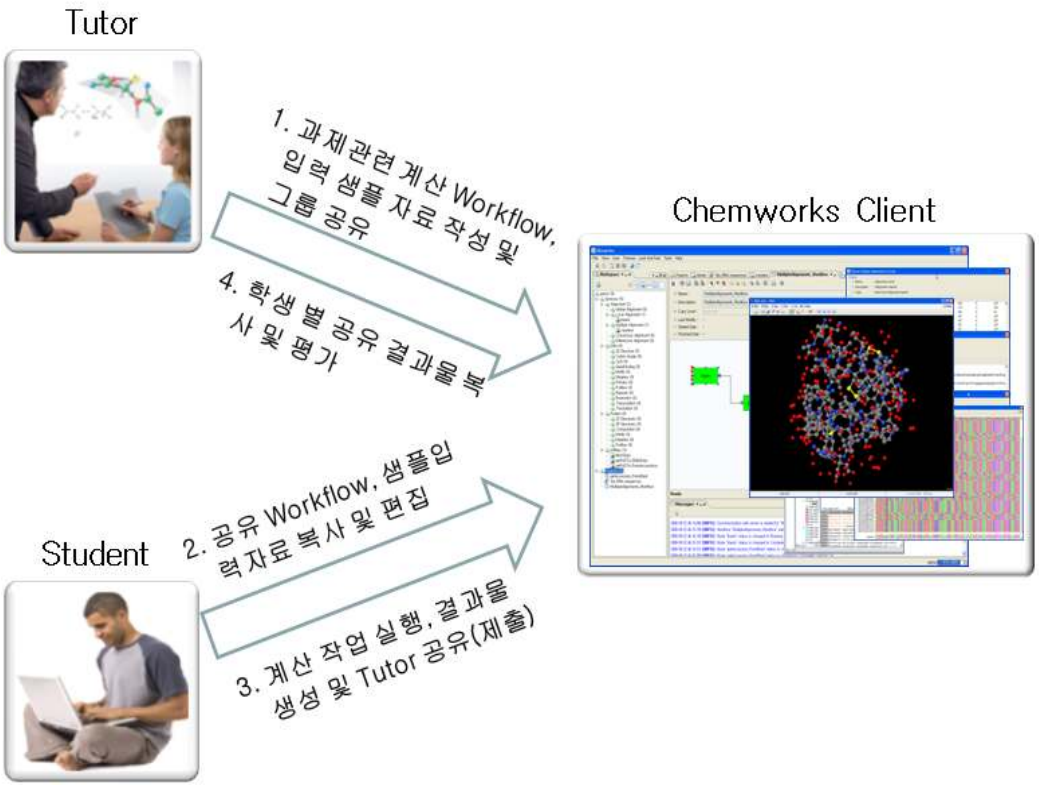

그림 1 계산화학 실습 시나리오

2.2. 요구 기능 구성

앞서 제시한 계산화학 실습 및 연구 환경에서의 문제점을 극복하고 실습 시나리오를 만족하기 위해서는 <표 2>에서 제시하는 바와 같은 주요 기능이 구현되어 져야 한다.

문제점	요구기능	비고
- 계산화학 도구 설치 및 유지보수가 어려움	- 별도의 프로그램 설치 없이 언제 어디서는 사용가능 하도록 구현 - Chemworks 서버 측에서 분석 도구 관리 및 자동 배포	클라이언트-서버 아키텍처
- UNIX 기반의 도구 사용의 어려움	- 일관성 있고 편리한 GUI 기반의 사용자 인터페이스 제공	
- 여러 단계의 계산 화학 분석파이프라인 구성이 어려움	- Drag&Drop 방식의 분석파이프라인(워크플로우) 편집 및 실행 자동화 기능 구현	
- 대규모 계산이 어려움	- 배치큐 연계를 통한 슈퍼컴퓨팅 기반 실행	다양한 배치큐 시스템 적용 가능
- 입출력 자료 및 계산 과정의 재사용이 어려움	- 사용자 별 입출력 자료, 분석파이프라인의 통합 workspace 관리 기능 제공	
- 사용자 간 자료 공유 및 협력 연구가 어려움	- 입출력자료, 분석파이프라인 공유/검색/복사 기능 구현	

표 2 Chemworks 주요 요구 기능

<그림 2>는 <표 2>의 주요 요구 기능을 포함한 Chemworks 시스템의 전체 기능 구성도이다.

그림 2 Chemworks 시스템 주요 기능 구성도

3. 적용 기술 분석 및 구현방법

3.1. 슈퍼컴퓨팅 기반 전체 시스템 구성도(클라이언트-서버)

Chemworks 시스템은 <그림 3>과 같이 웹서비시스 방식의 클라이언트-서버 아키텍처로 구성된다. 사용자는 시각화된 유저인터페이스인 Chemworks 클라이언트 프로그램을 통하여 계산화학 분석 파이프라인을 마치 레고 블록을 조립하듯 분석 도구의 입출력을 키워 맞추어 손쉽게 작성할 수 있다. 작성된 파이프라인은 Process와 Link로 구성된 XML 형태로 Chemworks 서버 측으로 전달되고 분석파이프라인에 포함된 각 실행 명령들은 배치 큐 서버에 배치 실행 작업으로 등록된다. 등록된 배치 작업은 워크플로우 엔진을 통하여 슈퍼컴퓨팅 서버 상에서 실행되고, 단계별 결과물은 사용자 데이터베이스에 저장된다. 사용자는 클라이언트 프로그램을 통하여 실시간으로 워크플로우 실행 현황과 단계별 결과물을 확인하고 분석할 수 있다.

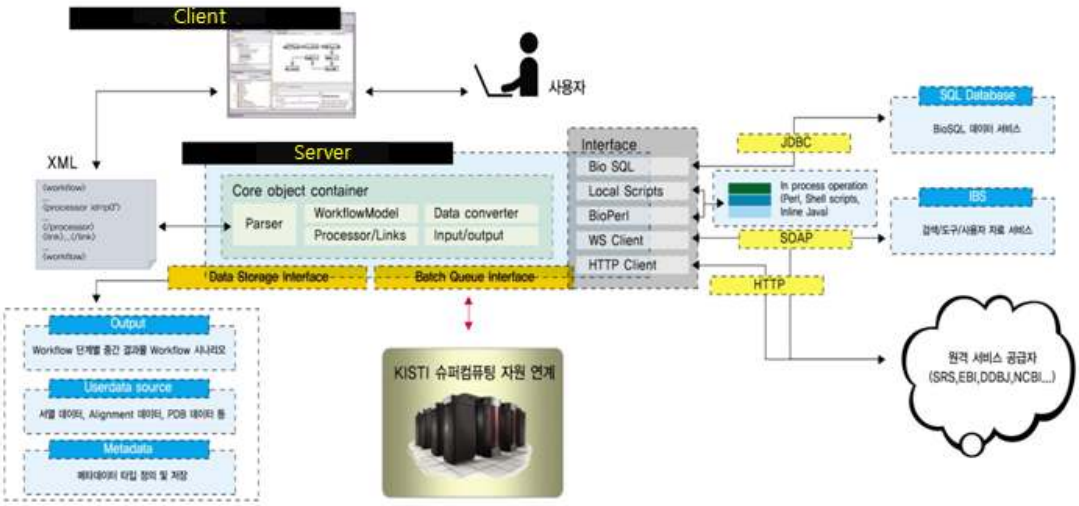

그림 3 슈퍼컴퓨팅 기반 전체 시스템 구성도

3.2. Spring framework

<그림 4>의 Chemworks 서버의 소프트웨어 아키텍처에서 보는 바와 같이 Chemworks 서버의 모든 객체 컴포넌트는 Spring framework에서 구동된다. Spring 프레임워크는 Rod Johnson의 “Expert One-on-One J2EE Design and Development"[2]에서 소개된 코드를 기초로 한 J2EE(Java 2 platform Enterprise Edition) 기반 애플리케이션 프레임워크 중 하나로, 객체의 라이프 사이클을 관리하기 위하여 Dependency Injection 기법을 사용하는 경량 컨테이너이다. Spring을 활용하면 복잡한 엔터프라이즈 애플리케이션 개발을 위해 EJB(Enterprise Java Bean)와 같은 복잡한 아키텍처를 사용하지 않고도 AOP(Aspect Oriented Programming), IoC(Inversion of Control), 테스트, 트랜잭션, 원격 서비스 연결, 객체-관계 맵핑, 보안, 웹 MVC(Model-View-Controller) 등과 같은 엔터프라이즈 자바 기술을 쉽게 적용할 수 있다. Spring의 핵심 기능인 IoC, 즉 제어 역행화는 Martin Fowler[3]에 따르면 객체간의 의존성 제어가 객체 코드에 의하지 않고 프레임워크에 의해 관리됨을 말한다. Spring은 이러한 IoC 개념을 기반으로 한 대표적인 IoC 컨테이너이다. IoC라는 용어가 직관적이 못하기 때문에 보다 이해하기 쉬운 Dependency Injection, 즉 의존성 삽입이라고도 말한다. Spring 프레임워크는 객체간의 의존성 삽입을 XML을 통하여 설정할 수 있도록 한다. 이렇게 함으로써 객체 간 의존성을 최소화하고 객체 컴포넌트의

모듈화를 극대화하여 매우 유연하고 확장성 높은 시스템을 구현할 수 있다.

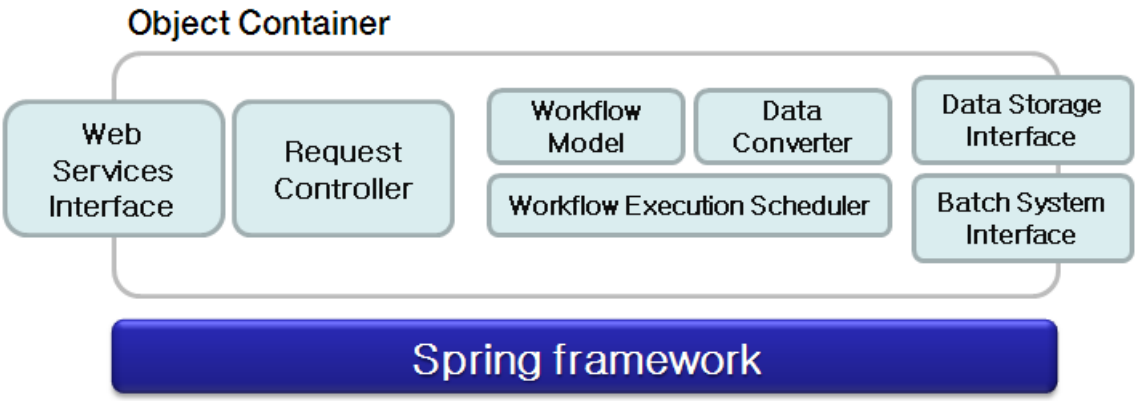

그림 4 Spring framework 기반 서버 아키텍처

3.3. 온톨로지 기반 데이터 양식 통합

온톨로지(Ontology)는 전산학 혹은 인공지능 분야에서 1970년대 후반 내지 1980년대 초반에서 사용죄기 시작하였고 그 이후 많은 개념상의 혼란이 있었다. 이에 1993년 Gruber가 “온톨로지는 어떤 관심분야에서의 개념의 정형화된 명세화이다(An ontology is a an explicit and formal specification of a conceptualization of a domain of interest)”[4]라는 말로 정의하였고, 이후 많은 학문 분야에서 받아들여져 응용되고 있다. 온톨로지에서의 지식은 클래스(class), 관계(relation), 함수(function), 공리(axion), 인스턴스(instance)의 다섯 가지 요소를 이용하여 형식화된다. 클래스는 일반적으로 개념어에 해당되며, 관계는 개념들을 규정하는 속성의 유형을 의미한다. 함수는 관계가 특정 값을 가질 때 성립되는 것이며, 공리는 논리의 전개나 추론의 근거가 되는 것으로 참으로 인정되는 문장을 의미한다. 그리고 인스턴스는 이와 같은 요소들이 결합되어진 실제의 값을 의미한다[5]. 온톨로지를 표현하기 위한 사용되는 표준 규약으로는 RDF, OWL, SWRL 등이 있다. RDF[6]는 XML에서 발전한 형태이며, subject, object, predicate로 이루어지며, 단순하게 개념 혹은 인스턴스 사이의 관계를 나타낸다. 일반적으로 복잡한 제약조건이 필요 없는 일반 응용 분야에서 RDF를 많이 사용한다. OWL[7]은 관계들 간의 계층적 구조, 관계 인스턴스 내에서의 논리적 제약조건 등을 포함한 언어이다. 정밀하고 논리적인 추론을 필요로 하는 경우에 사용한다. SWRL[8]은 추론을 위한 규칙을 정의하기 위하여 사용한다.

화학 분야에서의 온톨로지는 다양한 화학적 의미들을 표현하는 구조로 되어 있으며, 화학적 데이터의 의미를 효과적으로 해석할 수 있는 매우 중요한 기술로 인식되어 많은 세부 도메인에서 개발되고 사용되고 있다. 가장 대표적인 예로는 15,000 종의 화학적 엔티티를 생물학적 관점에서 정의하고 있는 ChEBI(Chemical Entities of Biological Interest)[9]가 있다. ChEBI는 EBI 생물학적 데이터베이스엣 사용되어 지는 화합물들에 대한 표준 규약을 정의하고 있다. EcoCyc 온톨로지[10]은 분자생물학과 생명정보 분야에서 이질적 정보 통합을 위해 정의되었는데 TAMBIS[11] 프로젝트에서 사용되어 지고 있다.

계산화학 분석 파이프라인 모델링을 위한 계산화학 데이터 양식에 대한 OWL 기반 온톨로지 표현을 <그림 5>과 같이 기술할 수 있다. 각각의 계산화학 분석도구에 대한 입출력 형식은 Gaussian, GAMESS Input, 분자구조 데이터 등과 같은 계산화학 데이터 일 수 도 있지만 문자열이나 숫자와 같은 Literal 형식일 수도 있다. 따라서 계산화학 분석 과정에 대한 워크플로우 모델링을 위한 OWL 표현에는 생명정보 데이터 양식과 Literal 형식을 모두 포함한다.

이렇게 정의된 OWL 표현 파일을 프로그램 코드에서 핸들링하기 위해서는 Jena[12]와 같은 온톨로지 처리 API가 필요하다. <그림 6>은 Jena API를 사용하여 정의된 OWL 파일을 파싱하여 계산화학 데이터 양식에 대한 계층적 트리 구조를 생성하고 Java 트리 뷰를 통해 보여주고 있다. Jena API를 사용하면 계산화학 분석도구 간의 입출력 연계 시 데이터 양식의 계층적 호환성을 쉽게 검사할 수 있다.

3.4. XML Configuration을 통한 계산화학 도구 통합

계산화학 분석 도구 통합을 위해 고려해야 할 점은 확장성과 유연성이다. 전 세계 계산 화학 분야 기업 또는 연구기관들은 각기 다른 형태의 포맷으로 계산화학 분석 도구들을 개발하고 배포해 왔기 때문에 이러한 이질적인 분석도구들의 통합을 위해서는 확장적이고 유연한 통합 스키마가 필요하다. XML은 이식성, 재사용성, 확장성, 효율적인 데이터 교환 등의 이점을 갖고 있어 이질적은 계산화학 분석 도구 통합을 위해 적합하다.

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns="http://www.bioworks.org/bioentry#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.bioworks.org/bioentry">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="EMBLFormatReport">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >EMBL format report</rdfs:label>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >EMBL format report</rdfs:comment>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="EMBLFormat"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ENZYMEEntry">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ENZYME Entry</rdfs:label>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="SwissProtFormatReport"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >ENZYME Entry</rdfs:comment>
  </owl:Class>
  <owl:Class rdf:ID="BtwistedReport">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >btwisted report</rdfs:comment>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >btwisted report</rdfs:label>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="GeneralReport"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="PFAMFormat">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >PFAM format</rdfs:label>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="DatabaseEntry"/>
    </rdfs:subClassOf>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >PFAM format</rdfs:comment>
  </owl:Class>
  ...
</red:RDF>
```

그림 5. 계산화학 분석 도구 입출력 양식을 위한 OWL 파일

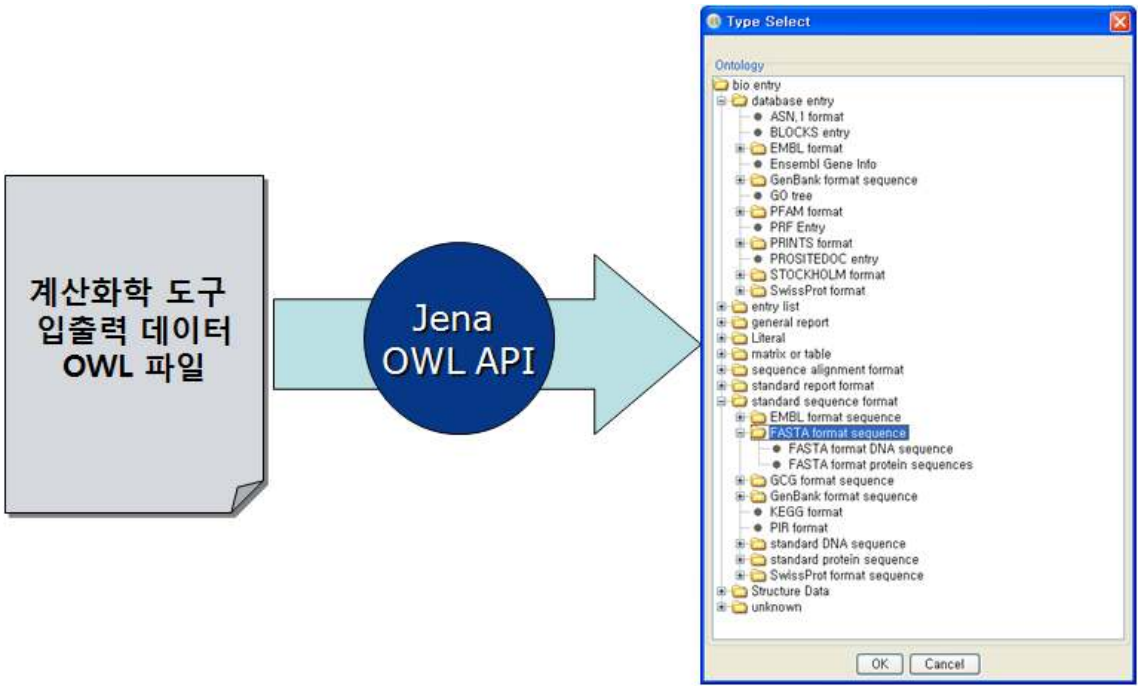

그림 6 Jena API를 통한 계산화학 도구 입출력 데이터 OWL 파싱 및 트리뷰

XML(eXtensible Markup Language)[13]은 데이터의 구조화, 저장, 상호교환을 위한 W3C의 공개 표준 규약으로서 그 이름으로부터 XML의 정의 및 특징을 말할 수 있다. 즉, Extensible은 사용자 정의 태그와 속성을 사용할 수 있어 추상화할 데이터 모델에 따라 마음대로 정의 가능하고, Markup은 중첩된 태그로 문서를 구조화하고 각 항목을 마크업 할 수 있다는 것이며, Language는 정해진 문법 규칙을 갖고 있다는 것이다. XML의 주요 장점을 살펴보면, 첫째 XML은 일반 텍스트 문서로서 플랫폼, 개발언어에 독립적이다. 따라서 시스템 또는 응용 프로그램 간의 상호 데이터 교환이 용이하다. 둘째 문서의 내용과 포맷팅을 분리하여 하나의 XML 문서를 다양한 형태로 Formatting 하거나 Display 할 수 있다. 셋째 XML은 확장적 모델링 언어로 확장 가능한 사용자 정의 태그를 통해 거의 모든 도메인의 개념을 추상화할 수 있다. 화학 분야에서도 분자 정보를 XML을 통해 마크업 하기 위한 CML(Chemical Markup Languate)[14]가 개발되어 다양한 계산화학 분석도구에서 입출력 데이터 양식으로 활용되어 지고 있다.

<그림 7>은 계산화학 분석도구 중 Gaussian을 통합하기 위한 XML 표현에 대한 하나의 예를 보여준다. 주목할 점은 각각의 입출력 데이터 타입을 계산화학 데이터 양식에 대한 OWL 표현 파일에서의 온톨로지 클래스 식별자

(OntClass URI)로 한다는 것이다.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
<!-- g09  -->

    <bean id="g09 " class="org.chemworks.domain.bioservice.CommandBioServiceDescriptor">
        <property name="name" value="g09"/>
        <property name ="description"><value>Gaussian 09</value></property>
        <property name ="shareLevel" value="1"/>
        <property name ="editable" value="true"/>
        <property name ="referenceLocation" value="g09.html"/>
        <property name ="available" value="true"/>
        <property name ="commandPath"><value>${chemworks.tool.path}/g09.run</value></property>
        <property name="inputDescriptors">
            <list>
                <bean class="org.chemworks.domain.bioservice.CommandBioServiceInputDescriptor">
                    <constructor-arg>
                        <value>input</value>
                    </constructor-arg>
                    <constructor-arg>
                        <value>${chemworks.ontclass.ns}#GaussianInput</value>
                    </constructor-arg>
                    <constructor-arg>
                        <ref bean="g09"/>
                    </constructor-arg>
                    <property name="description">
                        <value>Gaussian Input</value>
                    </property>
                    <property name="required" value="true"/>
                    <property name="valueSeparator" value=" "/>
                    <property name="nonValue" value="false"/>
                    <property name="asFileOnExecution" value="true"/>
                </bean>

            </list>
        </property>
        <property name="outputDescriptors">
            <list>
                <bean class="org.chemworks.domain.bioservice.CommandBioServiceOutputDescriptor">
                    <constructor-arg>
                        <value>output</value>
                    </constructor-arg>
                    <constructor-arg>
                        <value>${chemworks.ontclass.ns}#GaussianOutput</value>
                    </constructor-arg>
                    <constructor-arg>
                        <ref bean="g09"/>
                    </constructor-arg>
                    <property name="description">
                        <value>Output</value>
                    </property>
                </bean>
            </list>
        </property>
    </bean>
</beans>
```

그림 7 Gaussian 09에 대한 XML 정의

3.5. 계산화학 분석 파이프라인 실행 모델

<그림 8>은 하나의 계산화학 분석 파이프라인(워크플로우)의 실행 개념도이다. 하나의 워크플로우는 Process를 정점으로 하고 Process간 Link를 간선으로 하는 '방향성 그래프(Directed Graph)'로서 표현될 수 있다. 하나의 초기 실행 Process는 최상위 정점에 해당하는 Process가 되며 각각의 Process는 Link에 의해 연결되어 부모 Process와 자식 Process를 갖게 된다. 하나의 Process는 상위 Process들이 모두 종료되어 그것의 출력 값이 자식 Process의 입력 값으로 적합하게 설정되었을 때 병렬적으로 실행된다. 각각의 Process 실행 시에 지정된 계산화학 프로그램들이 호출되며 배치 큐 인터페이스를 통하여 슈퍼컴퓨팅 기반에서 실행된다. Process 간의 Link에 의한 데이터 전달이 일어나는 시점에서 특정 전이 조건에 대한 검사와 사용자가 정의한 스크립트 코드에 따른 데이터 변환이 이루어 질 수 있다.

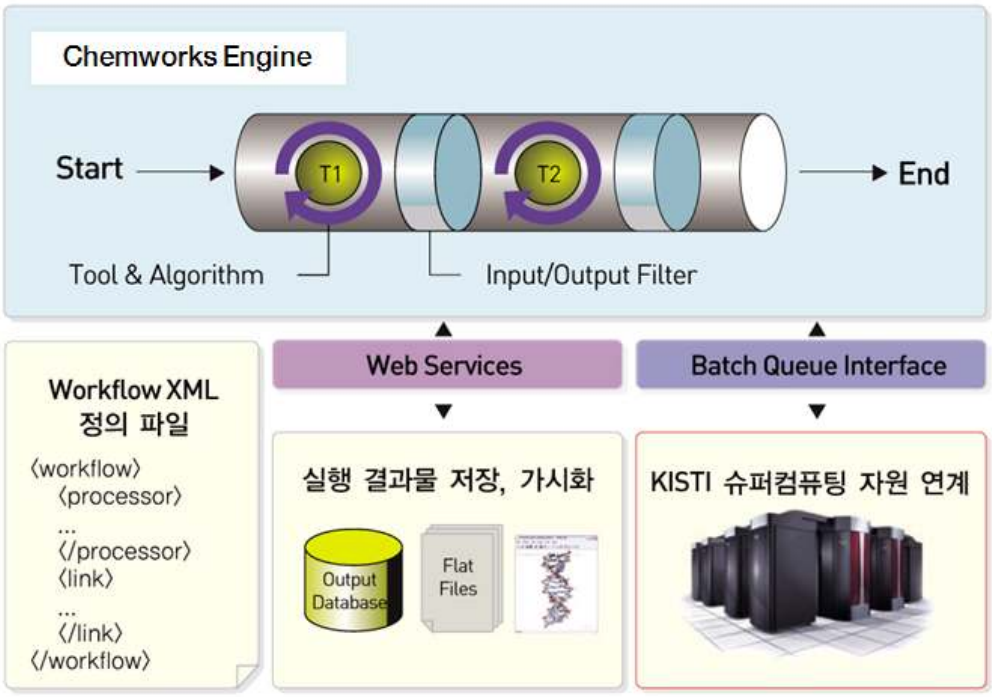

그림 8 계산화학 분석 파이프라인 실행 모델

3.6. 서버 푸시 방식의 배치큐 시스템 연계

앞서 제시한 바와 같이 Chemworks 서버 엔진은 실행 요청된 계산화학 분석 파이프라인의 각 분석 도구 들을 배치큐 시스템을 통해 슈퍼컴퓨팅 기반에서 실행토록 한다. 각각의 분석 도구 실행은 하나의 배치 작업으로 등록되어 병렬로 실행되고 실행 결과물은 데이터베이스에 적재된다. Chemworks 서버 엔진이 등록된 배치작업의 실행 상태를 동기화하기 위해서는 클라이언트 폴링 방식과 서버 푸시 방법이 있다. 전자의 경우는 Chemworks 서버 내에서 JobMonitor 쓰레드를 두고 주기적으로 배치큐 시스템에 특정 작업의 상태를 폴링하는 방식이다. 이 방식은 배치큐 시스템에 매번 상태 조회를 위한 쿼리를 요청하므로 조회할 작업 수가 많아질수록 배치큐 시스템의 부하를 초래할 수 있다. 게다가 배치큐 시스템의 종류에 따라 쿼리 방식이 다르므로 각 배치큐 시스템마다 연계 모듈을 재작성 해야 하는 단점이 있다. 후자의 경우 배치큐 시스템 측에서 특정 배치 작업의 상태가 변경 되었을 때 Chemworks 서버 측으로 데이터를 동기화하는 방식이다. 이 방식은 작업 상태가 변경되었을 때만 상태 동기화가 이루어지므로 배치큐 시스템의 부하를 최소화 할 수 있으며 거의 실시간으로 Chemworks 서버와 배치큐 시스템 간의 데이터 동기화를 수행할 수 있다. 게다가 다양한 배치큐 시스템에 거의 동일하게 적용이 가능하므로 별도의 배치큐 연계 모듈을 개발할 필요가 없다. <그림 9>은 배치큐 시스템 연계를 위한 두 가지 방식을 도식화 한 것이다.

Chemworks 시스템은 배치큐 시스템과 작업 상태 동기화를 위해 서버 푸시 방식을 도입한다. 다음은 배치 작업 등록 시에 각 실행 노드에서 실행하는 쉘 스크립트 코드이다.

```
#!/bin/sh
#
###Job name
#PBS -N bioworks_job
### Declare job non-rerunable
#PBS -r n
### Output files
#PBS -e /tmp
#PBS -o /tmp
#PBS -j oe
### Number of nodes (node property ev6 wanted)
#PBS -l nodes=1:blast:ppn=1
```

```
echo "Enter job_script.sh..."
echo "CMD==>$CMD"
echo "USER_NAME==>$USER_NAME"
echo "ACCT_NAME==>$ACCT_NAME"

WORKDIR=$WORKDIR/$PBS_JOBID

if [ ! -e $WORKDIR ]
then
        mkdir $WORKDIR
fi

cd $WORKDIR || exit 1

START_DATE=`date '+%Y%m%d%k%M%S'`

# Push job running status to bioworks server
JS_BASE_URL=http://150.183.158.234/bioworks/jobstatus.service
JS_STATUS=RUNNING

curl "$JS_BASE_URL?id=$PBS_JOBID&status=$JS_STATUS& ₩
                submittedDate=$SUBMIT_DATE&startedDate=$START_DATE"

# Run command
echo "Now run command: $CMD"

eval $CMD > $PBS_JOBID.out 2>&1

EXIT_CODE=$?

echo "Done. EXIT CODE: $EXIT_CODE"

FINISH_DATE=`date '+%Y%m%d%k%M%S'`

# Push job finished status to bioworks server
if [ $EXIT_CODE -eq 0 ]
then
        JS_STATUS=COMPLETED
else
```

```
        JS_STATUS=FAILED
fi

curl "$JS_BASE_URL?id=$PBS_JOBID&status=$JS_STATUS& ₩
        submittedDate=$SUBMIT_DATE& ₩
        startedDate=$START_DATE&finishedDate=$FINISH_DATE"

echo "Leave job_script.sh..."

exit $EXIT_CODE
```

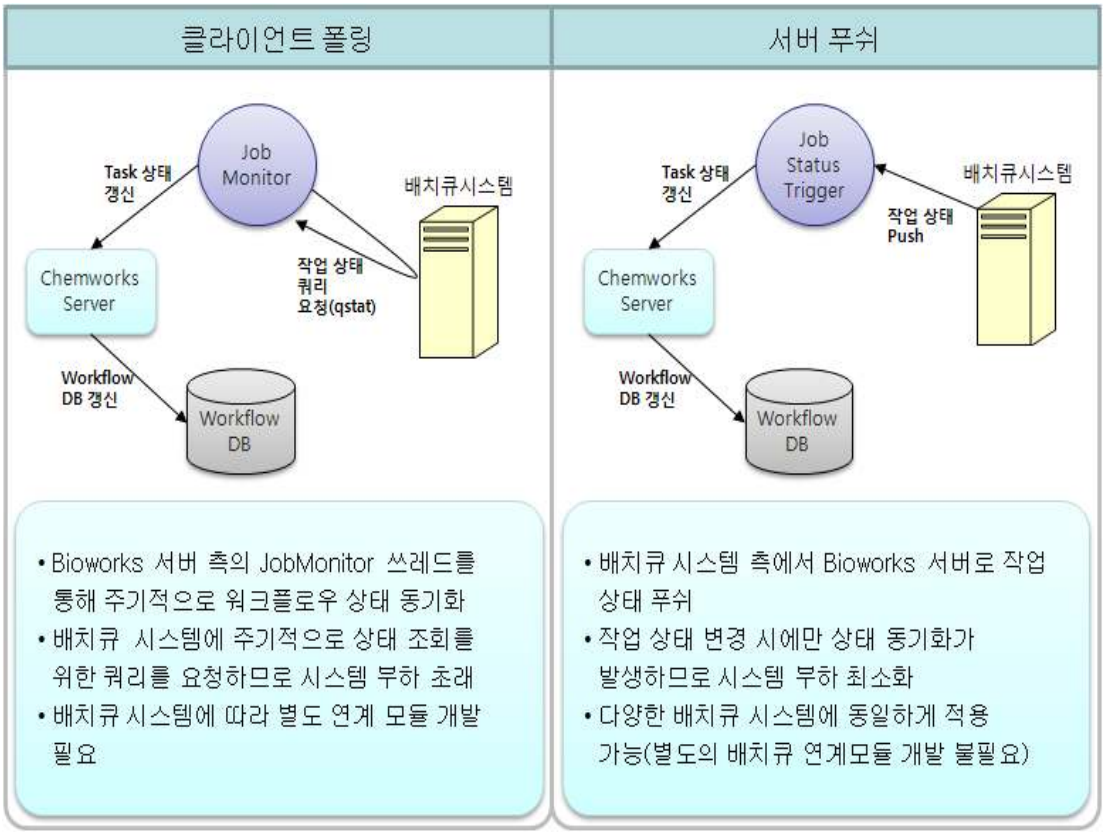

그림 9 Chemworks 서버와 배치큐 시스템 간의 작업 상태 동기화 방식

특정 명령이 실행되기 이전에 curl 명령을 통해 Chemworks 서버 측으로 해당 작업의 상태 정보가 전송된다. 배치큐 시스템으로부터 전송된 작업 상태정보를 Chemworks 서버에서 수신하기 위해서는 아래에서 보는 바와 같이 해당 배치 작업과 맵핑되는 분석 워크플로우 내 Task 상태정보를 변경하는

JobTrigger 모듈을 구현해야 한다.

```
import java.beans.PropertyEditorSupport;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.concurrent.CopyOnWriteArrayList;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.bqMonitor.JobDescriptor.JobStatus;
import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.web.HttpRequestHandler;
import org.springframework.web.bind.ServletRequestDataBinder;

public class HttpRequestJobStatusTrigger implements HttpRequestHandler,
JobStatusTrigger<HttpServletRequest> {
	private static final Log logger =
LogFactory.getLog(HttpRequestJobStatusTrigger.class);

	private static final PropertyEditorSupport statusPropertyEditor = new
PropertyEditorSupport() {
		public void setAsText(String text) throws
IllegalArgumentException {
			setValue(JobStatus.valueOf(text));
		}

		public String getAsText() {
			return String.valueOf(getValue());
		}
	};
	private static final CustomDateEditor datePropertyEditor = new
CustomDateEditor(new SimpleDateFormat("yyyyMMddHHmmss"), true);
```

```
	private List<JobStatusListener> listeners = new
CopyOnWriteArrayList<JobStatusListener>();

	public void handleRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		trigger(request);
	}

	public void trigger(HttpServletRequest source) {
		JobDescriptor jobDescriptor = new JobDescriptor();
		try {
			ServletRequestDataBinder binder =
createRequestDataBinder(jobDescriptor);

			binder.bind(source);

			logger.debug("Binded jobDescriptor==>" +
jobDescriptor);

			fireJobStatusChanged(new
JobStatusEvent(jobDescriptor));
		}
		catch (Exception e) {
			logger.error("Error in trigger:" + e);
		}
	}

	public void addJobStatusListener(JobStatusListener listener) {
		if (listener != null) {
			listeners.add(listener);
		}
	}

	public void removeJobStatusListener(JobStatusListener listener) {
		if (listener != null) {
			// To avoid the ConcurrentModificationException
			// for(Iterator<JobStatusListener> it =
listeners.iterator();
			// it.hasNext();){
```

```
                        // JobStatusListener cur = it.next();
                        // if(listener.equals(cur)){
                        // it.remove();
                        // }
                        // }
                        //
                        listeners.remove(listener);
                }

        }

        protected void fireJobStatusChanged(JobStatusEvent e) {
                if (!CollectionUtils.isEmpty(listeners)) {
                        for (JobStatusListener listener : listeners) {
                                if (listener != null) {
                                        listener.jobStatusChanged(e);
                                }
                        }
                }
        }

        private ServletRequestDataBinder createRequestDataBinder(JobDescriptor
jobDescriptor) {
                ServletRequestDataBinder binder = new
ServletRequestDataBinder(jobDescriptor);
                binder.registerCustomEditor(Date.class, datePropertyEditor);
                binder.registerCustomEditor(JobStatus.class,
statusPropertyEditor);
                return binder;
        }

}
```

3.7. 계산화학 분석 파이프라인 및 데이터 공유

계산화학이라는 말이 의미하듯이 계산화학 분석 과정은 화학과 IT 기술이 융합된 과정이다. 최적의 계산화학 분석과정을 모델링하고 이를 활용하기 위해서는 화학, 계산화학, IT 분야의 전문가 간의 협력 연구가 가장 효율적이

다. 서로 다른 분야의 연구자(또는 개발자)의 원활한 협력 연구를 위해서는 상호간의 커뮤니케이션을 위한 일관된 매개체, 즉 접점 인터페이스를 활용하는 것이 효과적이다. 이러한 점에서 계산화학 분석 파이프라인에 대한 워크플로우는 연구자 간 협업을 위한 최상의 접점 인터페이스라고 할 수 있다.

하나의 워크플로우를 연구자 간에 공유하는 방법은 워크플로우 게재, 검색, 복사의 순환 과정을 거친다. 워크플로우 게재 단계는 자신이 소유한 워크플로우에 대한 공유 수준(None, ReadOnly, ReadWrite)을 설정하여 다른 연구자가 복사하여 사용할 수 있도록 공개하는 것이다. 워크플로우 게재 단계에서는 다음의 몇 가지 사항을 고려하여 구현한다.

- 각 분석 도구의 입력 값 공유 여부: 워크플로우를 구성하는 각 분석 도구의 입력 값을 공유할 것인가를 설정할 수 있어야 한다. 왜냐하면 워크플로우 그 자체뿐만 아니라 각 입력 값도 매우 중요한 자료일 수 있기 때문이다.
- 사용자 또는 그룹 별 공유 수준 설정: 워크플로우를 게재하는 사용자의 해당 워크플로우에 대한 저작 권리를 보호하고 그룹 단위의 공동 작업이 가능토록 하기 위해서는 사용자 또는 그룹 별 공유 수준을 설정할 수 있어야 한다.

특정 사용자에 의해 워크플로우가 게재되면 다른 사용자가 이를 효과적으로 검색할 수 있게 하는 것도 매우 중요하다. 왜냐하면 초기에 계산화학 분석과정에 대한 워크플로우를 디자인 하는 일은 대부분의 사용자에게 있어 매우 부담스러운 작업일 수 있기 때문이다. 따라서 사용자가 작성하고자 하는 워크플로우와 유사한 공개 워크플로우를 간단한 검색 키워드만으로도 쉽고 빠르게 검색할 수 있는 기능을 구현해야 한다. 검색된 워크플로우를 복사하여 자신의 워크플로우로 재 작성하는 과정은 윈도우즈 탐색기에서 문서 파일을 복사하여 편집하는 과정과 유사하다. 복사된 워크플로우에 대한 사용자(또는 소유자)는 해당 워크플로우를 재편집하여 다시 다른 사용자에게 게재할 수 있다.

지금까지 제시한 워크플로우 공유방법은 하나의 동일한 기능을 수행하는 워크플로우의 복사물이 지나치게 많아져 자칫 서버 저장소의 부하를 초래할 수도 있다. 그러나 워크플로우의 공유 및 편집의 자유도가 증가하여 보다 많은 사용자가 초기 시드 워크플로우에 대한 활용이 많아질수록 계산화학 분석 과정에 보다 최적화된 워크플로우가 재생산될 가능성이 커진다. 이것은 특정 연구를 위한 계산화학 분석 파이프라인의 진화적 관점에서 보면 매우

효과적인 공유 방안이라 볼 수 있겠다.

4. 맺음말

계산화학은 분자동역학, 양자화학과 같은 이론화학에서의 수학적 방법론을 기초로 한 컴퓨터 프로그램을 사용하여 분자 구조 및 특성 등에 대한 실험적 사실을 증명하거나 예측하는 학문이다. 계산화학은 실험을 통해 확인하기 어려운, 또는 높은 비용을 요구로 하는 실험을 전산 모사하여 실험결과를 예측할 수 있어 매우 유용하게 활용되어 질 수 있는 분야이다. 그러나 컴퓨팅 능력의 기하급수적인 발달과 계산화학 기법의 발달에도 불구하고 계산화학 분석 도구가 실습 및 연구 환경에 활용되기에는 계산화학 분석 도구 설치 및 유지, Unix를 기반의 계산화학 프로그램의 사용 및 일관성 있는 UI와 관리도구의 부재, 여러 단계로 구성된 계산화학 분석인 경우 입출력 데이터 양식이 상이한 프로그램 간의 연계, 단백질 접힘 등과 같은 비교적 큰 분자에 대한 복잡한 계산, 사용자 간 자료 공유 및 협력 연구 등에서의 문제점들이 있다.

본 연구보고서에서는 이러한 문제점들을 극복하고 보다 효과적이고 확장적인 계산화학 실습 및 연구를 위한 슈퍼컴퓨팅 기반 통합 플랫폼(Chemworks) 구축 방안에 대해 제시하였다. Chemworks의 활용을 통하여 IT 기술이 부족한 연구자들이 보다 쉽게 계산화학 분석 파이프라인을 활용하여 연구생산성을 극대화 하고 Chemworks 기반의 연구자 간 협력연구를 통한 시너지 효과를 최대한 이끌어 낼 수 있을 것이라 기대된다.

참고문헌

1. ChemViz web page, http://www.shodor.org/chemviz/overview/ccbasics.html

2. Rod Johnson, *Expert One-on-One J2EE Design and Development*, Wrox book, ISBN: 978-0-7645-4385-2, 2002.

3. Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern*, http://martinfowler.com/articles/injection.html, 2004.

4. T. R. Gruber, *A translation approach to portable ontologies*, Knowledge Acquisition, 5(2):199-220, 1993.

5. 이재윤, *온톨로지 기반 과제관리 및 분석체제 구축 방안 연구*, 한국문헌정보학회지, 2006

6. W3C RDF web page, http://www.w3.org/RDF/

7. W3C OWL web page, http://www.w3.org/2004/OWL/

8. W3C SWRL web page, http://www.w3.org/Submission/SWRL/

9. Murray-Rust P., Rzepa, H. S., *Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles*, J. Chem. Inf. Comput. Sci. 39 (6): 928-942, doi:10.1021/ci990052b, 1999.

10. EcoCyc Ontology web page, http://ecocyc.PangeaSstems.com/ecocyc/ecocyc.html

11. Stevens R, et al., *TAMBIS: transparent access to multiple bioinformatics information sources.*, Bioinformatics. 16(2):184-5, 2000.

12. Jena web page, http://jena.sourceforge.net/

13.T. Bray et al., *Extensible Markup Language (XML) 1.0,* http://www.w3.org/TR/1998/REC-xml-19980210, 1998.

14.Murray-Rust P., Rzepa H. S., *Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles*, J. Chem. Inf. Comput. Sci. 39 (6): 928-942, 1999.