論文

XML을 이용한 객체지향 메타 모델링 기반 설계 프레임워크

주민식*, 최동훈**

Developement of a Object Oriented Based Meta Modeling Design Framework Using XML

Min Sik Chu* and Dong Hoon Choi**

ABSTRACT

Computer applications for engineering design evolve rapidly. Many design frameworks were developed by the simulation based systems so that organizations could achieve significant benefits due to cost reduction in designing. However, today's transient design issue requires being adaptable to more complicated and atypical problems. In this paper the Multidisciplinary Language Runtime (MLR) design framework is developed. The MLR provides flexible and extensible interface between analysis modules and numerical analysis codes. It also supports Meta Modeling, Meta Variable, and XML script for atypical design formulation. By applying object-oriented design scheme to implement abstractions of the key components required for iterative systems analyses, the MLR provides flexible and extensible problem-solving environment.

초 록

항공 설계분야에 있어서 최적설계는 방법론적인 요소와 방법론을 수행하는 소프트웨어인 설계프레임워크가 연구되어 왔다. 이러한 설계프레임워크는 효율적인 시스템으로 인식되어 많은 산업체 문제를 해결하여 왔지만 실제 산업체문제의 복잡성은 단일 최적화수준에서 다분야에 걸친 복잡한 비정형화된 최적설계 문제의 해결을 요구하고 있으므로 한계성을 지니고 있다. 따라서 본 연구에서는 복잡한 설계문제인 다분야통합최적설계문제를 위해서 유연성이 높은 새로운 설계문제 모델링 기반의 설계 프레임워크를 제안 하였다. 본 연구에서 제안하는 설계문제 모델링 방법은 기존의 설계프레임워크들이 절차적으로 생성하는 스크립트 기반의 모델링 기법이 아닌 설계문제 도메인의 형태에 맞추어 최적화 문제를 정의하는 시스템이다. 또한 시스템의 구조는 클라이언트 서버 구조가 아닌 P2P 구조의 시스템으로 개발되었으며 헬기 설계문제와 인공심장 문제를 적용하여 개발된 시스템의 유연성과 효율성을 보였다.

Key Words : XML(eXtensible Markup Language), Meta Modeling(메타 모델링), Multidisciplinary Design Optimization(다분야 통합 설계최적화)

† 2004년 11월 22일 접수 ~ 200년 2월 18일 심사완료

* 정회원, 한양대학교 기계설계학과 대학원

** 정회원, 한양대학교 기계설계학과

연락저자, E-mail : mdoframe@ihanyang.ac.kr

서울시 성동구 해당동 17 HIT 312호

Ⅰ. 서 론

항공 설계문제를 해결하는데 있어서 효과적으로 사용되는 설계 프레임워크는 설계문제에 소요되는

비용을 줄이는데 있어서 필수적인 요소이며 국내에서는 한양대학교 최적설계 신기술연구센터와 한국기계기술연구원을 위주로 MDO를 위한 범용적인 설계프레임워크를 개발하고 있으며 건국대 와 서울대의 항공관련 연구실에서는 항공기 설계 기반 프레임워크에 대한 연구가 진행되고 있다.

국외에서는 NASA나 기타 연구소에서 이러한 설계프레임워크 연구 및 개발에 주력을 하고 있다. 이러한 설계프레임워크는 협업 설계나 설계 대상 자체의 모델링을 위주로 하는 웹 기반 환경, 데이터베이스 기반, CAD 모듈과의 통합 시스템의 연구 분야와 실제 시뮬레이션 코드의 통합화와 통합된 시뮬레이션 코드를 이용한 효과적인 솔루션의 도출이라는 수치적 설계 분야로 양분되어 연구 되고 있다.

수치적 설계 연구 분야에 있어서는 iSIGHT(Engineous)[1], MODELCENTER(Phoenix Integration)[2], modeFRONTIER (ESTECO)[3], OPTIMUS(LMS)[4]등이 대표적인 설계프레임워크들이며 엔지니어링 관점의 모델링, 파싱 스크립트, 다양한 최적화 모듈, 클라이언트/서버 구조, 시뮬레이션 코드의 통합에 중점을 두고 있다.

위에서 언급한 상용 설계프레임워크들은 Giesing과 Barthelemy[5]가 분류한 MDO Elements (Design Formulation& Solution, Analysis Capabilities & Approximations, Information Management & Processing, Management & Cultural Implementation) 분류 중에 첫 번째를 제외한 나머지 세 요소들에 치중하고 있다. 하지만 현재의 설계문제는 일반적인 단일 최적화가 아닌 복잡다단한 형태의 다분야 통합 최적설계문제의 해결을 요구하고 있으며, 실제 산업체 설계문제의 형태는 매우 다양하며 이런 설계 문제는 기존의 설계프레임워크들로는 설정 및 모델링하기 힘든 경우가 있었다.

이는 개발된 설계프레임워크들은 모두 설계문제를 모델링하는 방식이 단일 프로세스 흐름의 정형화된 형태를 취하고 있기 때문이다. 다분야통합최적설계의 문제는 다중 단계의 복잡한 구조를 가지며 해석기와 최적화 기능간의 정적으로 분리된 클라이언트/서버 구조만으로 설계문제를 정의한다는 것은 어려운 점이 있었다.

이러한 문제는 설계프레임워크들의 개발 관점이 해석기 통합 중심 프레임워크로 개발된 시뮬레이션 기반 설계프레임워크라는 것과, Giesing과 Barthelemy가 첫 번째 요소로의 해결책으로 제시한 "설계 프레임워크의 유연성" 기능이 약하기 때문이다.

시뮬레이션 기반 설계프레임워크는 설계 도메인 내의 설계 자원들의 형태 및 위치에 따라 설계프레임워크를 배치시켜 문제를 설정하는 것이 아닌 설계프레임워크 환경에 맞게 설계문제를 정의하기 때문이다.

시뮬레이션 기반 프레임워크들의 또 다른 문제점은 최적화 모듈이나 근사화 모듈과 같은 개발자가 가지고 있는 수치해석 코드들의 통합화에 미흡하다는 것이다. 많은 상용 프레임워크들은 자체적인 외부 수치코드 통합 툴을 제공하고 있지만 개발자의 코드를 넣는데 있어서 시스템이 제공하는 툴의 규약이라는 한계성과 제약성을 가지고 있으므로 수치 해석 관련 코드의 통합이라는 관점에서는 어려움이 따른다.

프레임워크의 의미상 시뮬레이션 관련 설계 자원뿐 아니라 설계 관련 도메인 자원 전부를 재사용성의 극대화라는 면에서 기존의 설계프레임워크들의 기능은 매우 약하다.

이러한 설계 문제 도메인 자원의 재사용이라는 것은 단순한 시뮬레이션 코드뿐만 아니라 설계 자원 전반에 걸쳐서 적용되어야 하는 범위이며 설계 자원 자체와 설계문제 자체 까지도 객체화를 통한 재사용성의 극대화가 필요하다. 또한 비정형의 설계문제를 효율적으로 표현하기 위해서는 특정 언어를 사용하여 저장하고 검색하는 모델링 언어가 필수적이며 설계 문제 및 설계 자원에 대한 검색을 위해서는 설계문제의 내용을 구조적으로 설명하는 모델링 언어가 필요하다. 이런 언어는 플랫폼에 독립적이고, 설계자나 유저들에게 있어서 손쉽게 사용이 가능하며, 인터페이스가 유리해야 한다.

이러한 문제를 해결하기 위해서 기존에 연구된 선행연구로 개발된 분산 환경 기반 설계프레임워크[6]를 기반으로 연구를 진행하였으며 본 연구에서는 XML[7]을 이용한 객체지향 메타 모델링 기반의 MLR(Multidisciplinary Language Runtime)을 개발하였다. 기존의 설계프레임워크의 문제점인 기능상으로 분리된 클라이언트/서버 설계프레임워크 구조가 아닌 클라이언트와 서버 구조의 역할이 가변적으로 변하는 P2P(pear to pear) 구조이며, 모델링 언어로는 현재 표준화되고 있는 XML을 기반으로 하였다. 또한 설계문제에 객체 지향적인 구조를 도입하여 분석 정리하여 다분야통합설계최적화 및 복잡한 설계문제를 객체화 하여 기존의 시스템의 한계점을 극복하였다.

II. MLR

2.1 메타 모델링

비정형화된 문제에 대한 모델링을 하는 것으로 대표적인 것은 메타 모델링이다. 본 연구에서는 객체

표 1. MLR 객체를 이용한 다분야통합최적설계 문제 구성

MLR 객체 리스트				
Container Control Object(CCO)	하위 컴포넌트를 소유하고 있는 상위 객체		부모 객체	실제 객체 예
CCO 계열 객체	Optimization Object	header + Input parameter + Optimization Code+ output parameter	CCO	ADS, NLPQL, GA…
	Approximation Object	header + Input parameter + Approximation Code+ output parameter	CCO	Neural Network Code, RSM …
Common Object(CO)	Stand alone 형태로 동작이 가능한 객체			실제 객체 예
CO 객체 계열	Equation Object	header + Input parameter + equation + output parameter	CO	Methmetical Equation
	Applicaton Object	header + Input parameter(files) + application run + output parameter(files)	CO	ADAMS, ANSYS, NASTRAN…
* Model		CCO 객체와 CO객체로 조합된 설계문제 단위		ADS + ANSYS
MLR 객체로 조합한 다분야통합최적설계문제 구조				
단일/다분야	최적화 방법	객체 합성 구조		
Single Optimization				
	Optimization	Optimization Object(1) + CO		
Multidisciplinary Design Optimization	Multi Level Optimization			
	Multidisciplinary Design Optimization	MDF(Multidisciplinary Feasible)	Optimization Object(1) + CO(s)	
	Multidisciplinary Design Optimization	IDF(Individual Discipline Feasible)	Optimization Object(1) + CO(s)	
	Multidisciplinary Design Optimization	CO(Collaborative Optimization)	Optimization Object + Optimization Object Model(s)	
	Multidisciplinary Design Optimization	CSSO(Concurrent Subspace System Optimization)	Optimization Object + Approximation Object Model(s)	

기반으로 설계프로세스를 분류 했으며 이를 메타 모델링 기법을 적용하여 표현하였다. 메타(meta)란 어떤 개념적인 층위를 한 단계 올라선 것으로 의미하며 본 연구에서 메타 모델이란 최적설계문제 모델을 정의하기 위한 모델이라 정의 할 수가 있다. 메타 모델링 관점에서는 표현하고자 하는 대상 모델(최적설계문제)은 대체로 구조(structure)와 행위(behavior)로 분류되며 이러한 것은 "클래스, 컴포넌트 등이 어떻게 시스템을 구성하는가?"는 구조를 의미하며 "어떤 클래스의 인스턴스가 메시지를 받았을 때 어떻게 동작하는가?"는 행위를 의미하게 된다. 이런 메타 모델 클래스, 컴포넌트 관계를 구성하기 위한 설계문제의 구조와 행위를 정립한 것이 디자인 패턴이다.

디자인 패턴은 "특정한 상황에서 일반적 설계문제를 해결하기 위해 상호 교류하는 수정 가능한 클래스와 객체들"이라 정의한다.

디자인 패턴은 재사용 가능한 객체지향 설계를 만들기 위해 유용한 공통의 설계 구조로부터 중요 요소들을 식별하여 이들에게 적합한 이름을 선정하고 추상화 한 것이다. 이러한 디자인 패턴의 대표적인 기법은 클래스 상속과 객체 합성이다. 클래스 상속은 컴파일 시점 정적으로 정의 되고 프로그래밍 시에 직접 지원하므로 그대로 사용하게 되며, 구현을 쉽게 수정이 가능하며 서브 클래스는 모든 오퍼레이션이 아닌 일부만도 재정의 할 수가 있다. 하지만 바이너리 차원에서는 수정이 불가하며, 부모클래스의 수정시 서브 클래스도 변경을 하여야 한다. 객체 합성은 객체가 다른 객체에 대한 참조자를 얻는 방식으로 런타임시에 동적으로 이루어진다. 합성은 객체가 다른 객체의 인터페이스만을 바라보게 하기 때문에 객체간의 종속성이 줄어든다.

이 두 가지 개념을 설계프레임워크에 투영하였을 경우 객체 상속은 대표적으로 설계 개발자의 수치해석 코드를 새롭게 추가 할 경우에 MLR의 기존에 정의된 상위 클래스를 상속하여 시스템에 등록할 때 적합한 개념이다.

객체 합성 기법은 대표적으로 시뮬레이션 해석 코드와 최적화 모듈을 조합하여 더 상위객체(설계문제) 클래스를 만드는 목적에 적합하며 본 연구에서는 재귀적 합성(Recursive Composition) 기법을 도입하였다. 재귀적 합성은 단순한 것에서 복잡한

것을 점진적으로 만들어 내는 것으로 간단한 그래픽 요소들을 합성하여 문서를 만드는 방법과 같은 것이다. 최적설계문제 역시 이러한 객체 조합 개념의 형태로 표현할 수 있으며, 디자인 패턴에 의거하여 일반적인 설계문제에서 다분야통합 최적화 문제까지 분류하여 객체 분류 하였으며 분류된 객체들의 명세는 선행연구[6]에 분류 기준을 확장한 표 1과 같다.

2.2 시스템 디자인 구조

표 1에서 나타나듯이 최적설계문제를 디자인 패턴에 의거하여 분류한다면 대부분의 최적설계문제는 Common Object(CO)와 Container Control Object(CCO) 의 객체 합성으로 구성 된다. 그림 2 는 MLR시스템 내부를 역 엔지니어링한 UML[8]결과이다.

MLR의 모든 객체는 CMObject 클래스를 상속받는다. CMObject는 MLR내에서 가장 기본객체로서의 기능을 가진 클래스이며, 그림 1의 UML에서 나타나듯이 자기 자신을 참조하는 화살표 방향이 보이고 있다. 이는 재귀적 합성 기능에 의거해 자기 자신과 같은 타입의 객체를 내장할 수가 있음을 의미한다.

CMObject를 상속받는 클래스중 대표적인 것은 CContainerControlObject와 CCommonObject 클래스이며 앞의 접두어인 Container와 Common으로 모든 클래스 계열을 분류한다. CContainer 계열(상속 받은 클래스)의 클래스들은 객체 조합을 하는 부모 클래스이다. 의미대로 다른 객체들(상용 해석기, 수학식)을 내장하며 제어하는 하위 객체 관리 클래스로 객체를 조합, 관리 하는 객체이다. CContainer는 하위 객체들 간의 연성 관계를 정의하며 DSM (Design Structure Matrix)구조를 기반으로 객체들을 스케줄링 한다.

CCommonObject 클래스는 자체의 기능만으로 동작이 가능한 클래스들의 부모 클래스이다. CApplicatonObject는 대표적인 CCommonObject를 상속받은 자식 클래스로 상용해석 시뮬레이션 코드의 랩퍼 클래스이다. 그림 1에서 밝은 색의 클래스 객체들은 MLR에서 제공하는 클래스들이며 진한색의 CDesign_Optimization_ADSClass는 외부에서

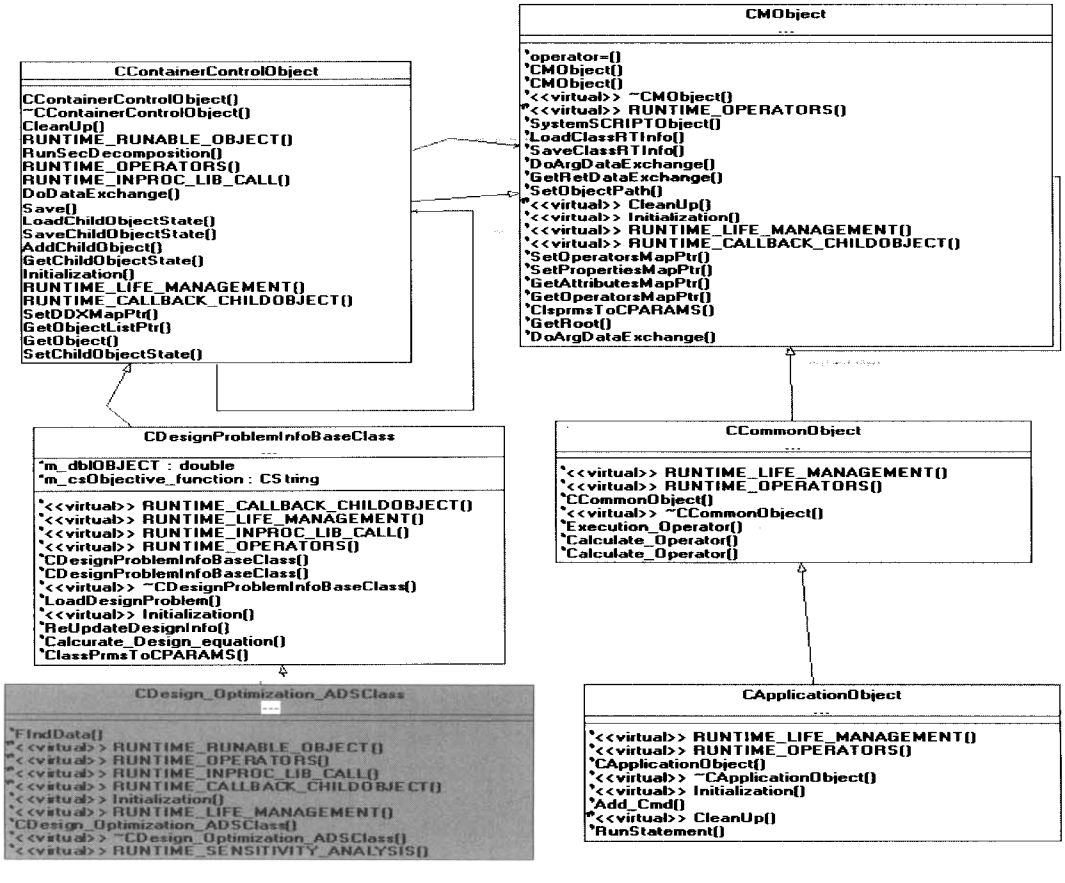

그림 1. MLR 코어의 역 엔지니어링

작성한 코드를 넣을 경우 생성한 클래스 예이다.

예를 들면 코드 개발자가 고유의 수치해석코드인 GA(유전자 알고리즘) 코드를 시스템에 삽입할시 CDesignProblemInfoBaseClass를 상속받은 후 가상함수[9]들 중에 필요한 부분만 인터페이스 코드를 작성하면 된다.

또한 MLR의 개발시 설계문제의 객체화라는 기능에 의거하여 표 2를 보면 여러 해석 프로세서를 표현하는 객체들과 최적화 객체의 조합으로 정의가 되어 있다. 이렇게 정의된 문제도 CContainerContol Object 라는 이름으로 객체(재귀적 객체합성)화되어 정의 되어 있으며 설계 문제역시 객체화 되어 외부에서 호출될 수가 있다.

설계문제의 객체화라는 점은 객체지향 방법론에 기준으로 정의 하였으므로 MLR내의 모든 객체는 동일한 인터페이스를 가지고 있다.

그림 1에서 보이듯이 CMObject로 상속받은 객체는 모두 통일된 인터페이스인 RUNTIME_LIFE_MANAGEMENT(), RUNTIME_ OPERATORS()로 정의 되어 있다. 이런 인터페이스에 의한 객체 추상화는 로컬 컴퓨터나 원격 컴퓨터에 배치된 설계 자원(해석관련 자원들)객체나 설계문제 객체 간에 무관하게 접근하여 제어와 접근을 할 수가 있다.

2.3 XML 기반 모델링

본 논문에서는 정의한 디자인 패턴 객체를 구조화하여 설계문제를 모델링하기 위해 XML을 사용하였다. XML의 사용 목적은 첫 번째는 데이터베이스로의 시스템의 전체적인 정보의 저장 및 검색의 역할의 편리성, 두 번째는 객체 정보 전송의 용이성으로 로컬 컴퓨터이던 원격 컴퓨터이던 간에 객체화된 정보를 다른 객체에 실시간으로 전달하며 이러한 기능은 이미 산업체의 SOAP[10]라는 표준으로 자리 잡고 있다는 점, 세 번째는 사용성의 편리성으로 XML은 기존의 프레임워크의 스크립트에 비해 매우 유저에게 익숙하며 널리 사용되는 확장 언어라는 점, 네 번째로는 객체표현기술의 타당성으로 마이크로 소프트나 기타 소프트웨어 산업체에서 객체 기술 언어로 XML이 사용 되고 있다.

마지막으로는 메타 모델링 언어의 적합성으로 본 논문에서 개발된 설계프레임워크는 비정형의 설계문제에 대해서 정의할 수가 있어야 한다. XML 문서는 문서 자체가 하나의 객체이며, 문서 안의 엘리먼트(element)가 독자적인 하나의 객체 단위로서 의미적으로 연관된 또 다른 객체를 포함 할 수도 있고, 문서의 구조가 계층적이라는 점에서 (비정형)설계 문제 표현의 논리적, 의미론적 구조화에 매우 적합하다.

표 2. XML로 정의한 헬기 설계문제

```
<ContainerControlObject RootID="MLR" MLRVersion="0.1" version="0.1"
xml:space="preserve">

<Object Classtype ="CDesign_Optimization_ADSClass" name ="MLR">
<Operators>
<Operator name="OBJ" ordor="-1">GW</Operator>
<Operator name="G1" ordor="-1" >HPREQ-95</Operator>
<Operator name="G2" ordor="-1" >EPNL-90</Operator>
<Operator name="G3" ordor="-1" >DOCS-14</Operator>
<Operator name="G4" ordor="-1" >ALFUSBR-0.035</Operator>
<Operator name="G5" ordor="-1" >-1*(ALFUSBR+0.15)</Operator>
<Operator name="G6" ordor="-1" >-1*(TAUTO-0.15)</Operator>
</Operators>

<Properties>3
<Property name="VT" TYPE ="DESVAR" LB="460" UB="560">25.2</Property>
<Property name="DL" TYPE ="DESVAR" LB="1.8" UB="2.8">45.1</Property>
<Property name="SGMA" TYPE ="DESVAR" LB="0.03" UB="0.05">30</Property>
<Property name="THETA1" TYPE ="DESVAR" LB="-5" UB="-10">30</Property>
<Property name="OBJ" TYPE= "OBJECTIVE">400</Property>
<Property name="G1" TYPE= "DCONSTR">0</Property>
<Property name="G2" TYPE= "DCONSTR">0</Property>
<Property name="G3" TYPE= "DCONSTR">0</Property>
<Property name="G4" TYPE= "DCONSTR">0</Property>
<Property name="G5" TYPE= "DCONSTR">0</Property>
<Property name="G6" TYPE= "DCONSTR">0</Property>
<Property name="GW">0.0</Property>
<Property name="HPREQ">0.0</Property>
<Property name="EPNL">0.0</Property>
<Property name="DOCS">0.0</Property>
<Property name="ALFUSBR">0.0</Property>
<Property name="TAUTO">0.0</Property>
<Property TYPE ="SYSTEM" name="ISTRAT">0</Property>
<Property TYPE ="SYSTEM" name="IOPT">4</Property>
<Property TYPE ="SYSTEM" name="IONED">7</Property>
</Properties>
</Object>

<Object name="INIT" Classtype="ApplicationObject">
<Properties>
<Property name="description">User Driver</Property>
<Property name="VT">20</Property>
<Property name="DL">51</Property>
<Property name="SGMA">50</Property>
<Property name="THETA1">50</Property>
<Property name="IAF">50</Property>
<Property name="IAF_OUT">0.0</Property>
<Property TYPE ="SYSTEM" name="IPADDRESS">166.104.125.174</Property>
<Property TYPE ="SYSTEM" name="PORT">6000</Property>
</Properties>
<Operators>
<Operator name="K1" input="X1 X2" output ="A1"></Operator>
</Operators>
</Object>

<Object name="DEVGEM" Classtype="ApplicationObject">
<Properties>
<Property name="description">User Driver</Property>
<Property name="IAF">20</Property>>
<Property name="IAF_OUT">0.0</Property>
<Property name="ARAHT">0.0</Property>
<Property name="ARHZTL">0.0</Property>
<Property name="ARAFIN">0.0</Property>
<Property name="X">0.0</Property>
<Property name="XTR">0.0</Property>
<Property name="NENG">0.0</Property>
<Property name="ENGC1">0.0</Property>
<Property name="ENGC2">0.0</Property>
<Property name="ENGC4">0.0</Property>
<Property name="ENGC5">0.0</Property>
<Property name="ENGC6">0.0</Property>
<Property name="ENGC7">0.0</Property>
<Property name="XALT">0.0</Property>
<Property name="XTP">0.0</Property>
<Property TYPE ="SYSTEM" name="IPADDRESS">166.104.125.174</Property>
<Property TYPE ="SYSTEM" name="PORT">6000</Property>
</Properties>
<Operators>
<Operator name="K1" input="X1 X2" output ="A1"></Operator>
</Operators>
</Object>


<Object name="CNTRLM" Classtype="ApplicationObject">
<Properties>
<Property name="description">User Driver</Property>
<Property name="X">20</Property>>
<Property name="XTR">0.0</Property>
<Property name="X_OUT">20</Property>>
```

```
<Property name="XTR_OUT">0.0</Property>

<Property TYPE ="SYSTEM" name="IPADDRESS">166.104.125.174</Property>
<Property TYPE ="SYSTEM" name="PORT">6000</Property>
</Properties>
<Operators>
<Operator name="K1" input="X1 X2" output ="A1"></Operator>
</Operators>
</Object>

.....

<DoDataExchange>
<LinkTo From="mlr" FROM_ID="VT" To="init" TO_ID="VT">0.00</LinkTo>
<LinkTo From="mlr" FROM_ID="DL" To="init" TO_ID="DL">0.00</LinkTo>
<LinkTo From="mlr" FROM_ID="SGMA" To="init" TO_ID="SGMA">0.00</LinkTo>
<LinkTo From="mlr" FROM_ID="THETA1" To="init"
TO_ID="THETA1">0.00</LinkTo>
<LinkTo From="mlr" FROM_ID="IAF" To="init" TO_ID="IAF">0.00</LinkTo>

<LinkTo From="sub_domain2" FROM_ID="HPREQ" To="mlr"
TO_ID="HPREQ">0.00</LinkTo>
<LinkTo From="noise" FROM_ID="EPNL" To="mlr" TO_ID="EPNL">0.00</LinkTo>
<LinkTo From="docs" FROM_ID="DOCS" To="mlr" TO_ID="DOCS">0.00</LinkTo>

.....

<LinkTo From="sub_domain2" FROM ID="HPREQ" To="hndqlt"
TO_ID="HPREQ">0.001</LinkTo>
<LinkTo From="sub_domain2" FROM_ID="OMGA" To="hndqlt"
TO_ID="OMGA">0.001</LinkTo>

</DoDataExchange>
</ContainerControlObject>
```

표 3. DEVGEM 객체 파일 랩핑 구현

```
xml version="1.0" encoding="utf-8" standalone="no" ?>
<ApplicationObject MLRVersion="0.1" type="MLR XML File" version="0.1">

<Object name="DEVGEM" type="ApplicationObject" BaseClass="CommonObject">

<SET_DESIGNINPUT_INFO
 INPUT_PATH="F:\E-DES\EngineeringDesignNurvSystem\MLR\DEVGEM\devgen.in"
MARKER_PATH="F:\E-DES\EngineeringDesignNurvSystem\MLR\DEVGEM\marker.in
"></SET_DESIGNINPUT_INFO>


<DVPREL id="IAF" TYPE="double" MARKER_ID="@@IAF@@" FORMAT="%10.3f" > 20
            </DVPREL>
<CMDLINE>
 START /WAIT /D F:\E-DES\EngineeringDesignNurvSystem\MLR\DEVGEM
devgem.exe </CMDLINE>

<SET_DESIGNOUTPUT_INFO
 FIND="" START_LINE="0"
END_LINE="18">F:\E-DES\EngineeringDesignNurvSystem\MLR\DEVGEM\devgem.ou
t</SET_DESIGNOUTPUT_INFO>

<DRESP ID="ARAHT" SEPERATOR ="" TYPE="double" LINE="0" START_PT="11"
END_PT="25"> 10 </DRESP>
<DRESP ID="ARHZTL" SEPERATOR ="" TYPE="double" LINE="1" START_PT="11"
END_PT="26"> 10 </DRESP>
<DRESP ID="ARAFIN" SEPERATOR ="" TYPE="double" LINE="2" START_PT="11"
END_PT="26"> 10 </DRESP>
<DRESP ID="X" SEPERATOR ="" TYPE="double" LINE="3" START_PT="8"
END_PT="21"> 10 </DRESP>
<DRESP ID="XTR" SEPERATOR ="" TYPE="double" LINE="4" START_PT="7"
END_PT="20"> 10 </DRESP>
<DRESP ID="NENG" SEPERATOR ="" TYPE="double" LINE="5" START PT="10"
END PT="20"> 10 </DRESP>
<DRESP ID="ENGC1" SEPERATOR ="" TYPE="double" LINE="6" START PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="ENGC2" SEPERATOR ="" TYPE="double" LINE="7" START_PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="ENGC4" SEPERATOR ="" TYPE="double" LINE="8" START_PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="ENGC5" SEPERATOR ="" TYPE="double" LINE="9" START_PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="ENGC6" SEPERATOR ="" TYPE="double" LINE="10" START_PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="ENGC7" SEPERATOR ="" TYPE="double" LINE="11" START_PT="10"
END_PT="25"> 10 </DRESP>
<DRESP ID="XALT" SEPERATOR ="" TYPE="double" LINE="16" START_PT="10"
END_PT="24"> 10 </DRESP>
<DRESP ID="XTP" SEPERATOR ="" TYPE="double" LINE="17" START PT="10"
END_PT="23"> 10 </DRESP>
<DRESP ID="IAF_OUT" SEPERATOR ="" TYPE="double" LINE="18" START_PT="12"
END_PT="19"> 10 </DRESP>
</Object>
</ApplicationObject>
```

표 2는 XML을 기반으로 헬기설계문제를 MLR의 객체 합성기법을 이용하여 모델링 한 것이다. 표 2의 XML스키마는 18개(표 2 에서는 몇 개의 객체만 표시)의 해석 객체들과 "MLR"이라는 이름을 가지는 최적화 객체를 정의하고 있으며 이들 객체간의 데이터 교환을 DoDataExchange라는 스키마로 정의하고 있다.

표 3은 표 2에서 모델링된 해석 객체 18개중 DEVGEM이라는 해석 객체의 파일 랩핑을 XML로 구현한 예를 보이고 있으며 크게 DVPREL이라는 입력 파싱 스키마와 CMDLINE 이라는 명령어 스키마 마지막으로 DRESP라는 출력 파일에서의 데이타 추출 스키마로 구현되어 있다.

2.4 객체 확장과 메타 변수

최적화 관련 수치해석 모듈을 넣기 위해서는 그림 1에서 나타나듯이 MLR에서 제공하는 클래스 중 상위 클래스인 CDesignProblemInfoBaseClass를 상속한다. 이 클래스는 그림 1에서 보여 주듯이 Container계열의 클래스이며 이 객체를 상속 받게 되면 가상함수들을 상속받아 인터페이스 코드만 작성하면 된다.

일반적으로 개발된 코드를 삽입할 경우 수치해석 개발자들은 내장할 설계프레임워크의 내부 규칙이나 규격에 따라 코드를 수정하거나 수치해석 코드인터페이스를 수정해야 하는 문제가 생긴다. 또한 CDesignProblemInfoBaseClass를 상속받아 클래스를 구현하더라도 설계자의 수치해석 코드의 입력 구조의 형식은 매우 많은 형태가 존재하므로 이러한 비정형 형태의 변수는 문자, 구조체, 객체일수도 있다.

이러한 비정형 변수를 인식하기 위해서 메타 변수라는 개념을 본 연구에서 도입 하였다.

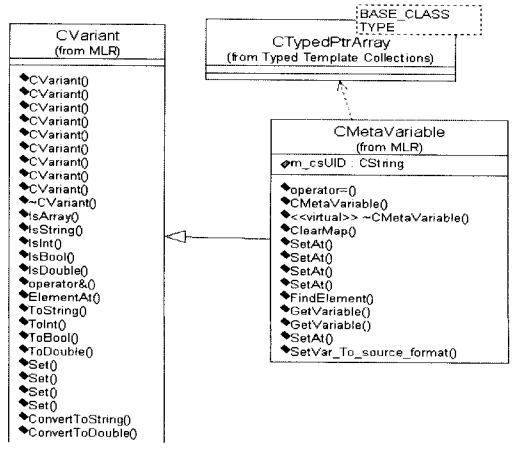

그림 2. MLR 내부의 메타 변수의 역 엔지니어링

XML은 어떠한 문서도 표현이 가능한 메타 모델 언어이며, XML로 표현된 비정형 변수는 그림 2에서 보여주는 것처럼 메타 변수라는 개념으로 시스템에 등록이 된다. 이런 메타 변수의 도입으로 시스템은 비정형 코드의 내장에 훨씬 더 유연해 진다.

2.5 적용 예제

본 문제의 첫 예제로는 헬기 설계문제로 한양대 최적설계신기술 연구센터와 미국 조지아공대 항공기 시스템설계 연구소의 공동 연구를 통해 예제로 개발된 모듈화된 프로그램으로 18개의 해석 모듈이 서로 연성되어 있다. 본 예제의 수행을 통하여 시스템의 유효성을 나타내고자 한다.

1)헬기설계문제

헬기설계문제의 그림 3와 같은 해석 프로세스간의 연관 관계를 가진 구조이며 총중량 최소화를 위한 문제이다. 구속조건으로는 필요한 축동력, 소음, 운영비, 기체의 받음각, 자동회전 개시시간을 설정치 이상으로 유지 하는 것으로 수식 (1)과 같다.

$$
\begin{aligned}
&\textit{Minimize}\\
&\quad \textit{gross weight}\\
&\textit{Subject to}\\
&\quad \textit{shaft power} < 95\\
&\quad \textit{noise} < 90\\
&\quad \textit{operating cost} < 14\\
&\quad -0.15 < \textit{attack angle} < 0.035\\
&\quad \textit{autorotation entry time} > 0.15\\
&\textit{Find}\\
&\quad 460 < \textit{tip speed} < 560\\
&\quad 1.8 < \textit{main rotor disk loading} < 2.8\\
&\quad 0.03 < \textit{main rotor solidity ratio} < 0.05\\
&\quad -10 < \textit{main rotor blad twist} < -5
\end{aligned}
\qquad (1)
$$

위와 같이 18개의 모듈로 구성된 설계문제를 구성한 뒤 최적화 모듈인 DOT[11]를 사용하여 MDF 방법을 이용하여 표 4와 같은 해를 구하였다. 수행결과 구속조건은 모두 만족하고 표 4의 데이터가 나타내듯이 목적함수는 대략 7% 줄어 들은 것을 알 수가 있다.

위의 헬기 설계문제는 여러 단계의 해석 프로세스를 가지고 있지만 해석 프로세스 각각은 수학 수식으로 구성되어 있으므로 최적 해를 구하는데 긴 설계시간이 필요로 하지 않는다.

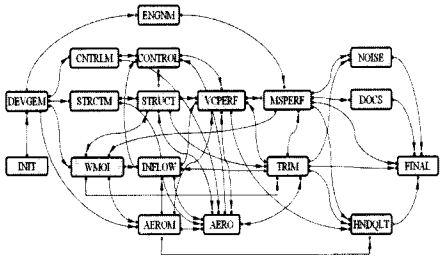

그림 3. 헬기 설계문제의 작업 흐름도

표 4. 최적설계를 수행한 결과

	gross weight	VT	DL	SGMA	THETA	Constraints
초기치	1495.6	460	2.7	0.04	-5.5	OK
MMFD	1401.8	464	2.21	0.044	-5.57	OK
SQP	1397.1	463	1.8	0.032	-9.89	OK

하지만 시간이 긴 해석단계가 중간 중간에 존재하고 해석 프로세스의 순서에 따라 피드백이 많이 존재한다면 최적 해를 구하는데 장시간이 소요되므로 이러한 피드백을 줄일 수 있는 분해기법의 도입이 필요로 하다.

개발된 설계프레임워크는 해석 프로세스를 메타 모델링 기법 이용하여 각각의 해석단계를 객체화 하였으므로 설계프레임워크로 하여금 설계구조행렬을 자동으로 생성 할 수가 있다.

표 1에 따르면 본 연구에서의 MDF 방법은 한 개의 최적화 객체와 여러 개의 일반객체들의 조합으로 구현이 된다. 헬기 설계문제는 하나의 최적화 객체와 17개의 해석 객체들 (일반객체를 상속받음 표 1과 그림 1 참조)이 조합하여 모델링 되었다. 17개 해석 객체(1개는 Final 단계로 자료

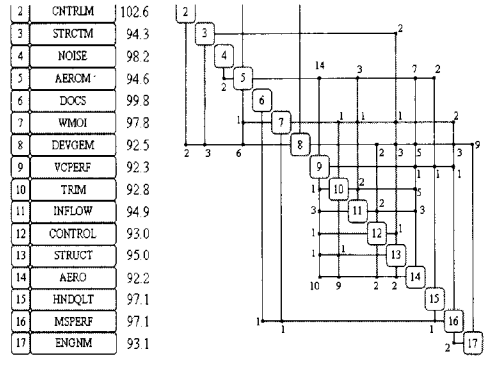

그림 4. 초기 헬기문제 설계프로세스

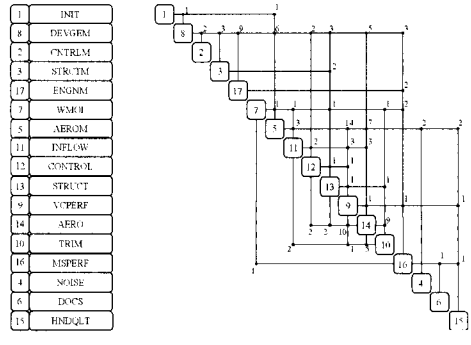

그림 5. 분해기법 도입 후 헬기 설계프로세스

를 모으는 역할만 하므로 설계구조행렬에서 제외)는 초기에는 그림 4와 같이 피드백이 49개가 존재한 형태로 구성이 되었으며 이를 시스템 분해기법을 도입하여 그림 5과 같은 피드백이 23개로 줄어들어 기존보다 해석프로세스 호출시간이 줄어든 구조임을 알 수가 있다.

2)인공심장문제

두 번째 예제는 NASA의 MDO 시험 문제인 Heart Dipole[12] 문제로 심장의 쌍극자 합 모멘트(resultant dipole moment)의 실험적 전해질 결정(experimental electrolytic determination)으로부터 공식화되었다. 일반적인 해석은 8개의 비선형 연립방정식의 해를 구하는 과정이다.

$$
\begin{aligned}
f_1 &= x_1 + x_2 - d_{mx} = 0 \\
f_2 &= x_3 + x_4 - d_{my} = 0 \\
f_3 &= x_5 x_1 + x_6 x_2 - x_7 x_3 - x_8 x_4 - d_A = 0 \\
f_4 &= x_7 x_1 + x_8 x_2 + x_5 x_3 + x_6 x_4 - d_B = 0 \\
f_5 &= x_1(x_5^2 - x_7^2) - 2x_5 x_7 x_7 + x_2(x_6^2 - x_8^2) - 2x_4 x_6 x_8 - d_C = 0 \\
f_6 &= x_5(x_5^2 - x_7^2) + 2x_1 x_5 x_7 + x_4(x_6^2 - x_8^2) - 2x_2 x_6 x_8 - d_D = 0 \\
f_7 &= x_1 x_5(x_5^2 - 3x_7^2) + x_5 x_7(x_7^2 - 3x_5^2) + x_2 x_6(x_6^2 - 3x_8^2) + x_4 x_8(x_8^2 - 2x_6^2) - d_E = 0 \\
f_8 &= x_5 x_5(x_5^2 - 3x_7^2) + x_1 x_7(x_7^2 - 3x_5^2) + x_4 x_6(x_6^2 - 3x_8^2) - x_2 x_8(x_8^2 - 2x_6^2) - d_F = 0
\end{aligned}
\tag{2}
$$

이 문제를 IDF와 MDF 방법론을 적용하여 구성할 때는 하위와 상위 레벨이 수식(3)(4)(5)와 같이 나누어진다.

IDF와 MDF와 다른 점은 서로간의 연성 변수가 등식구속조건으로 표현된다는 것이다.

$$
\begin{aligned}
&System \quad Optimizait \quad on\ (SO) \\
&\quad Find \qquad x_1, x_4, x_6, x_7, x_2^*, x_3^*, x_5^*, x_8^* \\
&\quad minimize \qquad f_5 + f_6 + f_7 + f_8 \\
&\quad subject \quad to \qquad f_i \geq 0, \quad i = 5,6,7,8 \\
&\qquad\qquad\qquad h_1 = (x_2 - x_2^*)^2 - (x_5 - x_5^*)^2 = 0 \\
&\qquad\qquad\qquad h_2 = (x_3 - x_3^*)^2 - (x_8 - x_8^*)^2 = 0
\end{aligned}
\tag{3}
$$

$$
\begin{aligned}
&Sub - system \ 1 \\
&\quad x_2 = 1 - x_1 \\
&\quad x_5 = \frac{1 - x_6 x_2 + x_7 x_3^* + x_8^* x_4}{x_1}
\end{aligned}
\tag{4}
$$

$$
\begin{aligned}
&Sub - system \ 2 \\
&\quad x_3 = 1 - x_4 \\
&\quad x_8 = \frac{1 - x_7 x_1 - x_5^* x_3 - x_6 x_4}{x_2^*}
\end{aligned}
\tag{5}
$$

CO는 각 단계에서 최적화를 하는 방법으로 상위에서는 목표 값(target value)을 지정하여 최적화 하는 방법이다. 수식(6)(7)(8)은 CO 방법론의 개념에 맞추어서 각 단계 최적화 수식을 나타내었으며 이에 따라 그림 6과 같이 컴퓨터 3대를 이용하여 문제를 설정하였다.

$$
\begin{aligned}
&Find \quad x_{1s}, x_{2s}, x_{3s}, x_{4s}, x_{5s}, x_{6s}, x_{7s}, x_{8s} \\
&to \ Minimize \quad f_5 + f_6 + f_7 + f_8 \\
&Subject \ to \quad f_j \geq 0, \quad j = 5,6,7,8 \\
&\quad j_1 = 0 \\
&\quad j_2 = 0
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
&Sub - system \ 1 \\
&Find \quad x_2, x_3, x_4, x_5, x_6, x_7 \\
&to \ Minimize \quad j_1 = (x_1 - x_{1s})^2 + (x_8 - x_{8s})^2 \\
&Subject \ to \quad f_j \geq 0, \quad j = 5,6,7,8
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
&Sub - system \ 2 \\
&Find \quad x_1, x_2, x_3, x_5, x_7, x_8 \\
&to \ Minimize \quad j_1 = (x_4 - x_{4s})^2 + (x_6 - x_{6s})^2 \\
&Subject \ to \quad f_j \geq 0, \quad j = 5,6,7,8
\end{aligned}
\tag{8}
$$

위의 수식(6)(7)(8)이 보여주는 CO 방법을 이용하여 기존의 설계프레임워크들로 구성했을 경우에는 그림 7과 같은 형태로 설정이 되며 시스템 레벨 컴퓨터에서 서브시스템 1, 2의 설계문제를 모두 가지고 있으며 이는 시스템 레벨과 서브시스템 레벨의 최적화 프로세스가 모두 한 컴퓨터에 집중되어 있다는 것을 의미한다. 대부분의 설계프레임워크들은 아래와 같은 구조를 취하고 있으며 클라이언트에 설계문제 집중적인 형태를 가지고 있다.

그림 7은 동일한 CO문제를, 서브시스템(1),(2)의 최적설계 부분에 2대의 컴퓨터가 추가하여 시스템 레벨의 컴퓨터에서 분리된 형태로 모델링 되었다.

간단한 문제의 경우 그림 6과 같이 설정 하여 중앙 집중적인 방식으로 해결하는 것이 편리하며 효과적이지만 서브시스템의 최적화 모듈이 시스템 부하를 많이 주거나, 솔루션을 도출하는데 필요한 코드가 고정된 특정 컴퓨터에만 설치되어 있는 경우에는 분산화된 그림7과 같은 서브시스템은 시스템레벨에서 분리된 형태가 필요 하다. 또한 다분야 통합최적설계의 방법론상 각각의 서브시스템들은 해당 고유의 도메인 영역을 가지고 있으며 이러한 해당 도메인은 해당 설계 팀에 분담되어 있는 경우에 설계문제를 분산화 하여 작업을 해야 하므로 협동 설계의

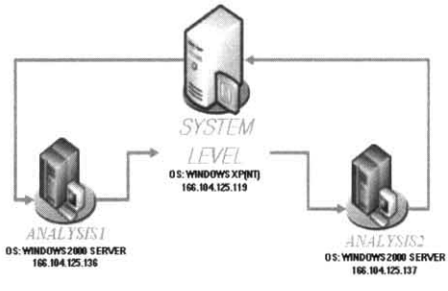

그림 6. 일반적인 컴퓨팅 환경 배치

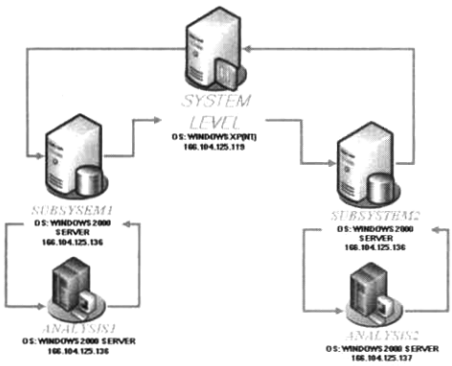

그림 7. CO 문제를 MLR 으로 배치할 경우

관점에서도 그림7과 같이 물리적으로 명확하게 분리되어야 효율적으로 분담업무가 가능하다.

아래의 표 5는 기존의 설계프레임워크로 그림 6의 형태를 이용하여 얻은 결과와 그림 7 형태로 구성하여 MLR로 얻은 결과를 나타낸 것으로 다른 설계 프레임워크와 비교하여 신뢰할 수 있는 솔루션을 얻었음을 알 수가 있다.

표 5. 헬기 설계문제의 MDO 방법론 적용 결과

Heart Dipole Problem			
	MDF	IDF	CO
Model Center			
Objective	-0.0006	0.00059	0.00461
Constraint Violation	0.00031 (f7)	0.0001(f7)	-0.00159 (f7)
iSIGHT			
Objective	0.03481	0.00217	-2.28592
Constraint Violation	-0.02417 (f8)	0.00087(f7)	0.54783 (f7)
MLR			
Objective	**0.00087**	**0.00074**	**0.00503**
Constraint Violation	**0.00024(f7)**	**0.00012(f7)**	**0.00021(f7)**

III. 결 론

일반적으로 복잡한 설계문제 및 다분야 통합최적설계 문제 구성하는 경우 발생하는 어려움을 해결하기 위해서 본 연구에서는 시뮬레이션 기반 설계 프레임워크의 문제점을 해결한 설계 자원의 위상이나 조건에 따라서 설계 프레임워크를 구성할 수 있는 설계 프레임워크를 개발하였다. 설계문제 자체도 객체지향 기법에 의하여 독립된 컴포넌트화 하였으며 객체를 분산화 및 재 사용성을 용이하게 하였다. 본 연구가 제안한 방법으로 정의된 설계문제는 객체 조합 구조를 이용하여 하위의 객체들을 구성하여 상위 객체를 구축할 수가 있다.

MLR 설계 프레임워크는 설계문제의 공통된 영역을 추상 클래스화 하여 설계자의 최적화 모듈이나 수치해석 코드를 삽입 시 주어진 가상함수만 상속받아 인터페이스만 정의하여 손쉽게 접속이 가능하다.

또한 XML기반의 설계 프레임워크이므로 시스템에서 외부로 데이터를 전송하거나 데이터의 수정 시 효과적인 구조를 가지고 있으므로 외부의 시스템과의 연동에 있어서 매우 효율적인 시스템으로 개발되었으며 실제 산업체 문제를 해결하는데 사용되어 많은 효과를 거두고 있다.

후 기

이 연구는 한국과학재단 지정 최적설계신기술 연구센터의 연구비 지원으로 수행되었습니다.

참고문헌

1) iSIGHT Developer's Guide Volume 2:MDOL Version 5.5- , Engineous Software Inc.

2) ModelCenter Technical White Paper, Improving The Engineering Process With Software Integration, Phoenix Integration Inc.

3) 인터넷 자료, modeFRONTIER, ESTECH, http://www.esteco.it.

4) 인터넷 자료, LMS Optimus, LMS International, http://www.lms.be.

5) Jossph P.Giesing(The Boeing Company), Jean-Francois M. Barthelemy(NASA Langley Research Center), A Summary of Inustry MDO Applications and Needs, Symposium on Multidisciplinary Analysis and Optimization, Sept.2-4, 1998.

6) 주민식, 최병렬, 이세정, 최동훈, "다분야 통합최적설계 지원 분산환경 프레임웍 개발", 항공우주학회.Proceedings of the KSAS Fall Annual Meeting 2001.

7) Beginning XML ,David Hunter외 5인 공저, Wrox Press LTD.

8) Grady Booch. Object-Oriented Analysis and Design with Applications, Benjamin/ Cummings, Redwood City, CA, 1994. Second Edition.

9) .NET Framework Essentials, 쑤안 타이, 호안램저, O'REILLY, 한빛 미디어.

10) 인터넷자료, Latest version of SOAP Version 1.2 specification, ttp://www.w3.org/TR/soap12.

11) Vanderplaats, G. N., Numerical Optimization Techniques for Engineering Design, 3rd Ed. VR&D,Inc., 1999.

12) Natalia Alexandrov and Srinivas Kodiyalam, "Initial Results of an MDO Method Evaluation Study, Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, Missouri, AIAA 98-4884, September 2-4, 1998.

13) A. O. Salas and J. C. Townsend, "Framework Requirements for MDO Application Development", 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, Missouri, AIAA 98-4740, September 2-4, 1998.

14) Elements of Reusable Object-Oriented Software Design Patterns, Erich Gamma, Richard Helm Ralph Johnson, John Vlissides, Addison-Wesley.

15) 이재우, 김종환, 정주영, 전권수, 변영환, "MDO 프레임워크 개발을 통한 항공기 날개 통합최적화 설계", 한국항공우주학회지, 제 32권, 6호, 2004.

16) 박형욱, 최동훈, 안병호, "다분야통합최적설계를 위한 적응분해기법", 한국항공우주학회지, 제 31권, 5호, 2003.

17) 황진용, 정주영, 이재우, 변영환, "다분야 통합환경에서의 데이터베이스 설계 연구", 한국항공우주학회지, 제 31권, 5호, 2003.