韓國經營科學會誌
第30권 第1號
2005年 3月

의사결정지원시스템에서 직관적이고 사용자 친숙한 모델 해결을 위한 모델과 솔버의 유연한 통합에 대한 연구*

이근우** · 허순영**

Flexible Integration of Models and Solvers for Intuitive and User-Friendly Model-Solution in Decision Support Systems*

Keun-Woo Lee** · Soon-Young Huh**

Abstract

Research in the decision sciences has continued to develop a variety of mathematical models as well as software tools supporting corporate decision-making. Yet, in spite of their potential usefulness, the models are little used in real-world decision making since the model solution processes are too complex for ordinary users to get accustomed. This paper proposes an intelligent and flexible model-solver integration framework that enables the user to solve decision problems using multiple models and solvers without having precise knowledge of the model-solution processes. Specifically, for intuitive model-solution, the framework enables a decision support system to suggest the compatible solvers of a model autonomously without direct user intervention and to solve the model by matching the model and solver parameters intelligently without any serious conflicts. Thus, the framework would improve the productivity of institutional model solving tasks by relieving the user from the burden of leaning model and solver semantics requiring considerable time and efforts.

Keyword : Decision Support Systems, Model Management, Model and Solver Integration

1. 개 요

하루가 다르게 변화하는 현대의 경영환경과 기업 간 경쟁의 심화는 기업에게 당면하는 많은 의사결정문제를 빠르고 정확하게 분석할 수 있는 능력을 요구한다. 그에 따라, 의사결정문제의 분석 및 해결을 지원하는 의사결정지원시스템(decision support system ; DSS)[32]의 역할이 더욱 중요해지고 있다.

논문접수일 : 2004년 9월 16일 논문게재확정일 : 2005년 1월 6일
* 본 연구는 대학 IT연구센터 육성 · 지원사업의 연구결과로 수행되었음.
** 한국과학기술원 테크노경영대학원

DSS 내에서 각각의 의사결정문제는 모델(model)로서 구조화되는데, 각 모델은, 사용자가 직관적으로 이해하고 실행하여 해결할 수 있도록 사용자가 쉽게 이해할 수 있는 모델링 언어를 이용하여 표현된다[16, 20].

기존의 DSS 연구는, 주로 경영과학(MS/OR) 문제를 중심으로 효과적인 모델링을 위한 모델링 언어에 대한 연구[9, 14, 17], 모델의 제작, 추출, 실행을 보다 쉽게 하기 위한 모델 표현 체계에 대한 연구[16, 20, 31, 34], 그리고, 모델의 재사용성을 높여 하나의 모델이 여러 다른 문제에 적용될 수 있도록 DSS 구성 요소(모델, 데이터, 사용자 인터페이스 등)간 효과적인 통합 방안에 대한 연구[5, 12, 30] 등을 중심으로 수행되었다. 그러나, 의사결정문제에 대한 해법 알고리즘인 솔버(solver)를 여러 모델 및 문제 데이터에 재사용하여 유연하게 의사결정을 지원하려는 연구는 많지 않았다. 솔버의 재사용은 모델과 솔버의 통합 관점에서 하나의 솔버가 유사한 구조를 가진 여러 모델에 적용될 수 있게 하고, 하나의 모델은 여러 솔버를 통해 다양한 문제 해결 목적에 따라 해결될 수 있게 함으로써 의사결정지원 과정의 사용자 편의성을 높이고 DSS의 구현과 유지를 보다 쉽게 할 수 있는 능률적인 시스템 아키텍처를 구현할 수 있게 한다.

이러한 모델과 솔버간 유연한 통합의 장점을 충분히 누리기 위해서는 사용자는 모델 및 모델을 구성하는 개별 문장들이 어떠한 의미를 갖는지, 그리고 자신의 문제 해결 목적 하에서 해당 모델에 대해 어떤 솔버들이 적용가능한지를 알고 있어야 한다. 또한, 적용 가능한 솔버가 선택되면 어떻게 각각의 모델 구성 문장 및 솔버 파라미터 값을 이용하여 자신의 문제 해결 목적에 가장 적절하게 모델을 풀 수 있는지 이해하고 있어야 한다[20]. 그러나, 보통의 DSS 사용자가 (종종 높은 지식수준의 전문 사용자도) 모델과 솔버의 의미를 충분히 이해하고 이를 풀어내는 것은 쉬운 일이 아니다. 이로 인해, 의사결정문제를 해결 하는데 있어 사용자는 기업의 전체 솔버 라이브러리로부터 현재 풀고자 하는 모델에 대해 적용 가능한 솔버를 적절히 골라내고 이를 적용하는데 많은 어려움을 겪는다[2, 13]. 이러한 어려움은 모델의 유용성을 인정하면서도 의사결정문제를 해결하는데 있어 모델의 사용을 주저하는 원인이 되고 있다. 의사결정문제에 대한 모델 사용의 이와 같은 어려움을 해소하기 위해 DSS는 다음과 같은 세가지 기능을 제공할 수 있어야 한다.

첫째, 해결하고자 하는 모델에 대해 구조적, 의미적으로 적용 가능한 솔버들을, 사용자의 개입 없이 자율적으로 제시할 수 있는 사용자 편의성을 가져야 한다. 이러한 자율적 솔버 제시(autonomous solver suggestion) 기능은 특정 모델에 대해 적용 가능한 솔버를 식별하기에 충분치 못한 지식을 가진 일반 사용자가 적절한 솔버를 쉽게 선택할 수 있도록 해주며, 부적절한 솔버 사용에 따른 오류를 방지할 수 있게 한다. 둘째, 특정 솔버가 선택될 때, 솔버가 요구하는 입력 파라미터 값을 모델로부터 제공할 수 있도록 모델과 솔버간 파라미터를 적절히 대응시킬 수 있어야 한다. 또한, 솔버의 계산 결과 값을 이에 대응되는 모델 데이터로서 전달할 수 있어야 한다. 즉, DSS는 모델과 솔버간 지능적인 파라미터 대응을 통해 사용자가 솔버 파라미터의 의미에 대한 충분한 지식이 없더라도 모델의 해를 구할 수 있도록 해야 한다(intelligent model solution). 셋째, DSS는 경영환경의 지속적인 변동 및 문제 해결 이론의 발전에 따른 모델과 솔버의 변화에 대해 유연하게 적응할 수 있는 아키텍처를 가져야 한다(adaptable system architecture). 구체적으로, 새롭게 제작되는 모델과 솔버가 추가될 때, 시스템의 큰 수정 없이도 기존 모델 및 솔버들과 바로 동작될 수 있도록 해야 한다. 이러한 변화에 대한 적응력은 많은 조직들이 내외부 네트워크상에 DSS를 분산시켜 구축함으로써[19, 21], 모델과 솔버의 변화가 보다 빈번이 발생됨에 따라 매우 필수적인 요구사항이 되고 있다.

DSS에게 요구되는 이러한 기능을 위해, 본 논문은 모델과 솔버의 변화에 적응하며, 자율적 솔버 제시 및 지능적 모델 해결 기능을 제공할 수 있는 모

델과 솔버의 통합 프레임웍을 제안한다. 이를 위해, 첫째, 포괄모델개념(generic model concept)[20]에 기반하여 모델과 솔버에 대한 체계적이고 통일된 표현 체계를 제시한다. 이러한 표현체계는 모델링 관행 및 적용 도메인에 관계없이 다양한 형태의 모델과 솔버를 모두 수용할 수 있으며, 따라서 제안 프레임웍은 다양한 도메인의 DSS에 쉽게 적용될 수 있을 것이다. 둘째, 모델 솔버 표현 체계를 기반으로 다중 에이전트 기술(multi agent technology)을 이용한 시스템 아키텍처를 제시한다. 에이전트는 서로 통신하고 사용자와 능동적으로 상호작용하여 특정 모델에 대해 적용 가능한 솔버를 제시하고 이를 모델에 적절히 대응시킴으로써 사용자의 모델 해결을 돕는다[1, 4]. 에이전트 협업(collaboration)에 기반한 이러한 다중 에이전트 접근법은 이질적인 모델과 솔버의 상호작용을 조정하고 이들간 발생할 수 있는 의미적, 구조적 충돌을 해결할 수 있는 유연한 시스템 플랫폼을 제공하는데 있어 매우 효과적이다[23, 24].

본 논문의 구성은 다음과 같다. 2장에서는 DSS에서 모델과 솔버의 통합에 관한 기존의 연구들을 간략히 살펴보고, 3장에서는 금융 DSS의 예를 통해 포괄모델개념 기반의 모델, 솔버 및 그들간 상호작용에 대한 표현 구조를 정의한다. 특히, 모델과 솔버간 표준 인터페이스 정의를 어렵게 하는 모델링 충돌 이슈에 중점을 두어 설명한다. 4장에서는 객체지향 패러다임을 이용하여 제안 프레임웍을 위한 물리적 시스템 아키텍처를 제시하고, 자율적 솔버 제시와 지능적 모델 해결의 구체적 과정을 설명한다. 또한, 이를 기반으로 구현한 원형시스템의 구조 및 실행 화면 예를 제시한다. 5장에서는 제안 프레임웍의 의의와 한계 및 그에 따른 향후 연구 방향을 제시하며, 6장에서 본 논문을 요약함으로써 결론을 맺는다.

2. 문헌 연구

모델과 솔버의 통합에 관한 기존 연구들은, 매개 파일이나 데이터베이스 테이블 등을 이용하여 모델과 솔버간 데이터를 교환하는 느슨한 통합(loose integration) 방식과 데이터 교환 및 솔버 실행을 위해 내부의 연산 및 서브루틴을 이용하는 견고한 통합(tight integration) 방식의 두 가지 접근법 중 하나를 기반으로 하고 있다[18].

첫째, 느슨한 통합 방식에서는 MPS 포맷[32]과 같은 표준화된 형식의 매개 파일을 이용하여 모델과 솔버간 파라미터를 교환한다[14, 15, 17]. 즉, 모델은 매개 파일의 표준 포맷에 따라 변환되고 솔버는 다시 해당 포맷에 맞추어 이를 읽어 들임으로써 계산에 필요한 입력 파라미터 값을 얻는다. 이러한 느슨한 통합 방식은 최적화 모델(optimization models)을 대상으로 하는 많은 연구와 소프트웨어 패키지에서 광범위하게 선택되고 구현되었는데, 이는 최적화 모델이 결정변수(decision variable), 제약식(constraints), 목적함수(objective function) 등 사전에 정의된 수식 단위 블록 집합으로 일관되게 표현될 수 있기 때문이다. 이러한 최적화 모델의 일관된 블록 구조는 매개 텍스트 파일의 생성 및 모델로부터의 변환을 쉽게 표준화하여 구현할 수 있게 하며, 솔버가 해당 블록의 의미를 이해할 필요 없이 수학적으로 해석하고 실행할 수 있도록 한다. 따라서, 느슨한 통합 방식에서는 모델을 읽고 표준 매개 파일을 생성하는 프로그램을 제작함으로써 모델과 솔버간 인터페이스를 완전 자동화할 수 있다[16]. 또한, 느슨하게 통합된 모델과 솔버는 상호간에 독립적이기 때문에, 새로운 모델 및 솔버를 추가하거나 기존의 것을 변경하는 것이 용이하다. 그러나, 느슨한 통합 방식의 이러한 장점은 전형적인 모델 구조를 갖는 최적화 모델에만 적용될 수 있다는 한계를 갖는다. 즉, 전형적 모델 구조를 갖지 않는 비최적화 모델에 대해서는 매개 파일이 표준화되기 어렵고 모든 개별적 모델에 맞게 달라져야 하기 때문에 이러한 느슨한 통합 방식의 적용이 어렵게 된다.

둘째, 견고한 통합 방식은 DSS 내에 하드코딩된 고유의 연산이나 서브루틴을 통해 모델과 솔버의 인터페이스를 구현한다[20, 27]. 하드코딩된 인

터페이스 연산은 대부분 컴퓨터의 주메모리를 이용하며 하드디스크와 같은 2차 메모리는 거의 사용하지 않기 때문에 느슨한 통합 방식에 비해 실행속도가 빠르다. 또한, 각 인터페이스 연산은 개별 모델 및 솔버의 구조적, 의미적 특성에 따라 개별적으로 커스터다이즈 될 수 있기 때문에 느슨한 통합 방식과 달리 특정 모델 구조에 대한 제약이 없으며, 따라서 매우 다양한 도메인 및 모델링 패러다임에 적용 가능하다. 이와 같이 견고하게 결합된 모델과 솔버간 인터페이스를 통해, 견고한 통합 방식은 보다 정교하고 믿을만한 모델의 해를 제공할 수 있다. 그러나, 하드코딩된 인터페이스 연산은 새로운 모델이나 솔버가 추가되거나 기존의 것이 변경될 때마다 이를 반영하기 위해서는 전체 시스템이 수정되어야 한다는 점에서 시스템의 유지보수를 어렵게 한다.

이상에서 살펴본 바와 같이, 기존의 두가지 모델과 솔버의 통합방식은 각자의 한계로 인해 다양한 도메인에서 실제적인 유용성을 제공하기 어려운 것으로 판단된다. 이러한 기존 방식의 한계를 극복하기 위해, 본 논문에서는 기존 방식이 갖는 장점을 최대한 수용하며 동시에 사용자에게 충분한 실용성을 제공할 수 있는 모델과 솔버간 통합 프레임웍을 제안한다. 구체적으로, 제안 프레임웍은 느슨한 통합 방식을 기반으로 견고한 통합 방식에서와 같은 인터페이스 연산의 개별 모델 및 솔버에 대한 커스터마이즈가 가능하도록 개발된다. 인터페이스 연산의 커스터마이즈는 모델과 솔버가 각기 다른 모델링 패러다임을 기반으로 상이한 구조를 갖더라도 상호연동 가능하도록 하여 다양한 도메인에 적용될 수 있도록 할 것이다. 이를 위한 모델과 솔버간 매개체로서, 필요한 인터페이스 파라미터를 관리하고 커스터마이즈된 인터페이스 연산에 대한 지침을 제공하는 사전 형태의 정보등록소를 정의한다. 이때, 정보등록소는 기존의 매개 파일 기반 느슨한 통합 방식에서와 달리 특정 모델 구조에 제약 받지 않는 일반적 구조를 갖도록 함으로써 특정 도메인에 종속되지 않도록 한다. 또한, 솔버 제시 및 모델 해결 기능을 수행하는 주체로서 모델에이전트와 솔버에이전트를 정의하고, 이들이 정보등록소를 기반으로 자율성 및 의사교환 능력을 가지고 해당 기능을 수행하도록 한다. 정보등록소 내의 인터페이스 지침과 에이전트에 의한 실행 어플리케이션을 분리하는 이러한 접근법은 또한 전체 시스템의 수정 없이 정보등록소의 갱신만으로 모델과 솔버간 인터페이스를 쉽게 변경할 수 있도록 함으로써 견고한 통합 방식에서 발생되었던 인터페이스 커스터마이즈에 따른 유지보수 문제를 완화시킨다. 따라서, 제안 프레임웍은 모델과 솔버의 독립성을 확보하며, 동시에 견고한 통합 방식에서와 같은 다양한 모델 구조에 대한 일반성 및 신뢰성의 장점을 갖는 것을 목표로 한다.

3. 포괄모델개념을 이용한 모델과 솔버의 표현구조

이 장에서는 모델간 발생 가능한 모델링 충돌(modeling conflict) 이슈에 중점을 두고 본 논문에서 정의하는 모델과 솔버 및 그들간 상호작용에 대한 구조적이고 통일된 표현방식에 대해 설명한다. 우선, 제안하는 표현방식의 주요 개념에 대해 설명하고, 의미충돌(semantic conflict), 이름충돌(naming conflict), 구조충돌(structural conflict)의 세가지 모델링 충돌을 정의한다. 또한, 보다 실제적인 설명을 위해 금융 DSS의 옵션가격결정 예를 이용한다.

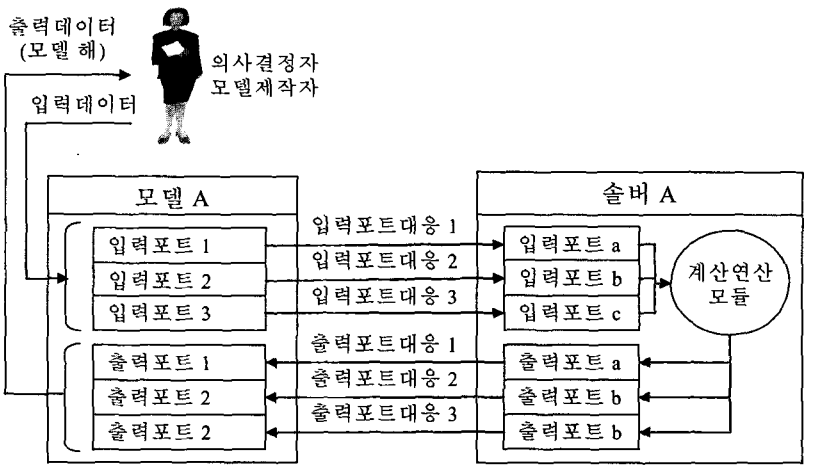
3.1 모델, 솔버 표현구조의 주요 구성체

본 논문에서는 모델 솔버 표현구조의 개념 프레임웍으로서 포괄모델개념[20]을 선택하고, 이를 확장하여 솔버의 선택 및 적용이 보다 유연하게 이루어지도록 한다. 본래의 포괄모델개념에서 솔버는 모델에 포함되어 있는 형태로 정의되었으나, 이는 솔버의 구현과 모델의 구현이 뒤섞이게 함으로써, 모델을 이해하고 유지하는 것을 어렵게 하며, 신규 솔버의 추가나 기존 솔버의 변경을 힘들게 한다. 따

라서, 본 논문에서는 솔버를 모델로부터 분리하여 개별적인 독립 구성체로서 정의한다.

먼저, **모델**은 의사결정문제를 표현하는 객체로서 정의되며, 문제의 의미를 정의하는 서술적 구성 문장들의 유지 및 갱신을 담당한다. 이를 위해 모델은 고유한 이름과 함께 포트집합을 자신의 속성으로 갖는데, 각 **포트(port)**는 모델의 구성 문장과 일대일 대응 관계를 갖는 객체로서 전체적으로 모델의 외부 인터페이스를 구성한다. 모델은 포트를 통해 모델 관리자나 솔버, 데이터베이스 등의 외부 환경과 상호작용한다. 이때 모델의 포트는 데이터를 받아들이는 입력포트(input port ; inport)와 문제 해결 결과를 외부 환경에게 제공하는 출력포트(output port ; outport)의 두 가지 종류로 구분된다. 각 포트는 모델의 개별 문장이 갖는 데이터 값 또는 수식을 표현하기 위한 속성 및 연산 등을 갖는다.

한편, **솔버**는 모델로 표현된 의사결정문제의 해를 계산하는 역할을 담당한다. 솔버의 속성으로는 고유 이름, 모델 포트와 상호작용하기 위한 포트 집합, 그리고 해결 알고리즘을 구현한 계산 연산 등이다. 이때, 솔버포트 또한 입력포트와 출력포트로 구분되는데 입력포트는 계산 연산의 입력 파라미터 값을 받아들이며 출력포트는 계산 결과값(모델의 해)을 제공한다. 이를 위해 솔버의 입력포트는 모델의 입력포트와, 솔버의 출력포트는 모델의 출력포트와 대응된다[그림 1]. 즉, 모델을 풀기 위한 솔버가 선택되면, 모델은 자신의 입력포트 값을 대응하는 솔버의 입력포트에 전달한다. 솔버는 계산 연산을 이용하여 출력포트 값을 생성하고 이를 모델의 출력포트에 되돌려준다. 이와 같이, 모델을 풀기 위해서는 모델의 각 포트들을 솔버의 포트에 적절히 대응시켜주는 것이 필요하다. 본 논문에서는 이러한 포트간 대응 관계를 **포트대응(port mapping)**이라 부른다.



[그림 1] 포괄모델개념에 기반한 모델, 솔버 및 상호작용 구조

포트대응은 개별 모델과 각 모델의 적용 가능 솔버간 사전에 정의됨으로써 모델과 솔버간 종속관계를 유지하고 상호작용하여 문제를 해결하도록 할 수 있다. 그러나, 솔버의 관점에서 모델 사이에 의미적 혹은 구조적인 충돌이 발생될 수 있기 때문에[3, 8, 29], 포트대응은 각 개별 모델 및 솔버에 대해 커스터마이즈되어 그러한 충돌을 충분히 해소시킬 수 있는 형태로 정의되어야 한다. 일반적으로 다음과 같은 세가지 종류의 모델링 충돌이 있을 수 있다.

- **의미충돌**은 개별 모델이 가지는 문제의 의미에 따라 솔버가 참조해야 하는 모델포트의 종류가 달라지는 것을 의미한다. 예를 들어 특정 솔버를 공유하는 생산비용 최적화 모델과 상품 외판원

의 최단 판매경로 탐색 모델의 두 최적화 모델이 있을 때, 해당 솔버는 각 모델에서 서로 다른 입력포트 값을 참조하여 해를 구해야 한다. 즉, 생산비용 최적화 모델에서는 생산비용이 솔버의 입력포트 값이 되나, 최단 판매경로 탐색 모델에서는 판매처간 거리가 입력포트 값으로 사용된다. 결국, 문제의 특정 의미를 내포하는 모델포트를 그러한 의미가 없는 수학적 변수 형태인 솔버포트에 대응함으로써 이러한 의미충돌이 발생하게 된다.

- **이름충돌**은 모델이 동일한 포트에 대해 유사동의어를 사용하거나 또는 서로 다른 포트에 대해 동음이의어를 사용하는 경우에 발생한다. 예를 들어 생산비용 최적화 모델에서 생산공장에 대한 포트는 "plant"라 명명될 수 있으나 또 다른 생산비용 최적화 모델에서는 "factory"로 이름지어질 수 있다. 이 경우, "plant"와 "factory"는 유사동의어이다. 한편, 어떤 모델에서는 생산량에 대한 포트이름으로서 "quantities"를 사용하고 다른 모델에서는 이를 원재료량에 대한 포트이름으로서 사용할 수 있으며, 이 경우 두 "quantities"는 동음이의어이다.
- **구조충돌**은 개별 모델에서 동일 포트가 서로 다른 데이터 포맷으로 표현되는 경우에 발생한다. 예를 들어, 한 모델에서 생산 비용은 생산량에 따른 변동 비용 값들의 집합으로서 표현될 수 있으나 다른 모델에서는 이들의 평균 비용을 나타내는 스칼라 값으로서 표시될 수 있다.

DSS는 의미충돌 및 이름충돌에 관계없이 개별 모델과 해당 모델에 적용 가능한 솔버간 대응되어야 할 포트조합을 정확히 파악할 수 있어야 하고, 각 포트대응간 구조충돌이 있을 경우에는 포트간 데이터 포맷을 적절히 변환할 수 있어야 한다. 본 논문에서는 이러한 포트대응 및 데이터 포맷 변환 방법을 통틀어 **인터페이스 규칙(interfacing rule)**이라 부른다. 각 모델과 솔버간 인터페이스 규칙이 적절히 정의되면, 이를 통해 DSS는 모델 포트의 의미적, 구조적 이질성에 의해 발생되는 세가지 모델링 충돌을 해소하면서 모델과 솔버를 적절히 연결하여 모델의 해를 구할 수 있을 것이다.

다음 절에서는 이러한 모델링 충돌의 발생 가능성 및 모델과 솔버간 상호작용에 대해 금융 DSS의 옵션 가격결정 예를 통해 보다 현실적인 관점에서 살펴보기로 한다.

3.2 옵션가격결정 예

2장에서 언급한 바와 같이 모델, 솔버 통합에 대한 대부분의 기존 연구는 최적화 모델을 중심으로 이루어졌다. 최적화 모델은 사전에 정의된 전형적 수식 블록 집합으로 표현될 수 있기 때문에 이들간에는 앞서 언급한 세가지 모델링 충돌이 거의 발생하지 않으며, 따라서 모델과 솔버간 상호작용은 상대적으로 쉽게 구현될 수 있었다. 그러나, 본 논문은 그러한 표준화된 블록 구조를 갖지 않는 비최적화 모델에도 적용 가능한 범용의 모델과 솔버의 통합 프레임웍을 개발하는 것이 목적이므로 금융 DSS의 예를 통하여 비최적화 관점에서 제안 프레임웍을 설명하고자 한다.

금융 DSS는 순현재가치(net present value ; NPV)[22]와 같은 금융상품에 대한 분석치를 계산하여 제공함으로써 사용자의 금융상품 평가를 지원한다. 이때, 각 금융 상품은 분석치 계산의 편의성을 위하여 해당 상품의 속성과 계산하고자 하는 분석치들에 대한 문장들의 집합으로 표현된 모델로서 구조화된다. 또한, 이러한 상품 모델에 대한 솔버로서 다수의 가격결정알고리즘이 제공된다. 여기서는 상품 모델의 예로서 주식옵션과 통화옵션의 두가지 상품을 고려한다. 주식옵션은 그 소유자에게 옵션의 기초자산(underlying product)으로 지정된 특정 주식을 사전에 정해진 가격으로 미래의 특정 시점에 매입하거나 매도할 수 있는 권리를 부여하는 상품이며, 통화옵션은 유사한 조건으로 외환을 매입하거나 매도할 수 있는 권리를 부여하는 상품이다. [그림 2]는 주식옵션에 대한 모델을 구조모델링언

어(structured modeling language ; SML)[17]를 이용하여 표현한 예를 보여준다. SML에서 모델은 특정 속성에 대한 문장을 의미하는 포트(정식 명칭은 genera)의 집합으로 표현되며, 의미적으로 관련된 포트들은 상위 개념단위인 모듈(module)로서 그룹화된다.

[그림 2]에서 STOCK_OPTION으로 명명된 주식옵션모델은 OPTION_DATA, STOCK_DATA, VALUATION_RESULTS의 세가지 모듈로 구성된다. OPTION_DATA는 옵션 고유의 속성에 대한 포트를 그룹화한 것이며, STOCK_DATA는 옵션의 기초자산인 주식에 대한 포트를, VALUATION_RESULTS는 옵션가치평가를 위한 분석치에 대한 포트를 그룹화한 것이다. 그러한 분석치 중 본 논문에서는 NPV만을 고려하기 때문에, VALUATION_RESULTS는 NPV 포트만을 갖는다. 한편, 통화옵션모델 역시 OPTION_DATA, CURRENCY_DATA, VALUATION_RESULTS의 세가지 모듈로 구성되며, CURRENCY_DATA가 옵션의 기초자산인 외환에 대한 포트를 그룹화한다. 이들 모델의 목적은 각각 NPV값을 계산하는 것이므로, 두 모델 모두 NPV 포트가 출력포트이며, 나머지 포트는 NPV값을 계산하기 위해 필요한 데이터로서 사용자나 외부 데이터원천으로부터 그 값을 공급받는 입력포트이다.

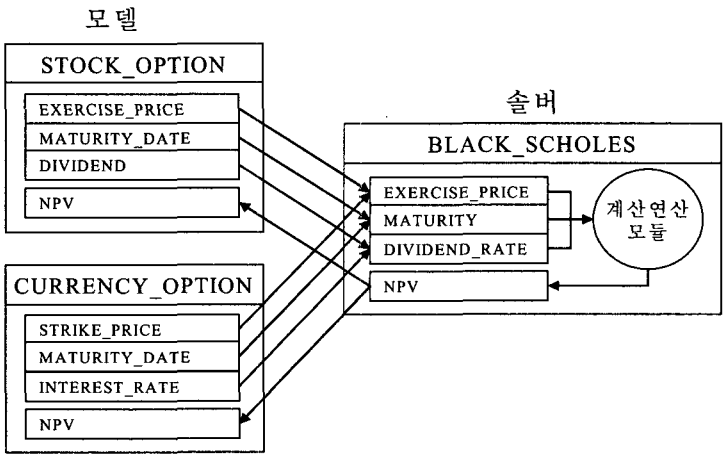
```
&OPTION_DATA  OPTION_DATA for properties pertinent to option products
  OPTION (STOCK) /ce/  There is an OPTION on STOCK.
  OPTION_TYPE (OPTION) /a/  OPTION_TYPE indicates whether OPTION is call or put.
  EXERCISE_TYPE (OPTION) /a/  EXERCISE_TYPE indicates whether OPTION is European or American.
  EXERCISE_PRICE (OPTION) /a/  EXERCISE_PRICE is the prearranged price at which the holder of OPTION can buy or sell STOCK.
  MATURITY_DATE (OPTION) /a/  MATURITY_DATE is the date at or by which the holder of OPTION can buy or sell STOCK.
  OPTION_PRICE (OPTION) /va/ : Real+  OPTION has a current OPTION_PRICE in a financial market.
&STOCK_DATA  STOCK_DATA for properties pertinent to stock products
  STOCK /pe/  There is a STOCK in a financial market.
  STOCK_PRICE (STOCK) /va/ : Real+  STOCK has a current STOCK_PRICE in a financial market.
  S_PRICE_VOL (STOCK_PRICE) /va/ : 0 < Real < 1  STOCK_PRICE has a STOCK_PRICE_VOLATILITY.
  DIV_PAY_DATEi (STOCK) /a/ : Integer >= 19000101  STOCK has a list of DIVIDEND_PAYING_DATE.
  DIVIDEND (DIV_PAY_DATEi) /a/ : Real+  At each DIVIDEND_PAYING_DATE, there is a DIVIDEND in $.
&VALUATION_RESULTS  VALUATION_RESULTS for evaluating OPTION
  NPV (OPTION) /va/ : Real+  OPTION has a NET_PRESENT_VALUE.
```

[그림 2] STOCK_OPTION에 대한 SML 모델

한편, 이 두 모델에 대한 솔버의 예로서 본 논문에서는 대표적 옵션가격결정 알고리즘인 블랙숄즈 방법(Black-Scholes method)[22]을 고려한다. 블랙숄즈방법 솔버는 옵션의 NPV를 계산하기 위해 행사가격(exercise price), 만기일(maturity date), 배당률(dividend rate) 등의 세가지 입력포트 값을 필요로 한다. 첫째, 행사가격은 옵션 소유자가 기초자산을 매입 혹은 매도할 수 있는 사전에 정의된 가격으로서, exercise price 혹은 strike price라 불린다. 둘째, 만기일은 옵션소유자가 해당일에 혹은 해당일까지 기초자산을 매입 혹은 매도할 수 있는 날짜를 의미한다. 보통 만기일은 특정 날짜로 표현되거나 현재일로부터 만기일까지의 시간간격으로 표현되는데, 블랙숄즈방법 솔버에서는 후자의 시간간격을 이용한다. 셋째, 배당률은 주식옵션의 기초자산 주식에서 지급되는 배당의 연간 비율이다. 그러나, 실제로 주식의 배당은 불규칙적으로 지급되기 때문에 블랙숄즈방법 솔버를 사용하기 위해서는 불규칙적인 배당을 연간 비율로 변환하는 것이 필요하다. 한편, 통화옵션의 경우에는 주식이 배당을 지급하는 것과 같이 기초자산 통화가 이자를 지급하므로, 기초자산 통화의 이자율이 주식옵션에서의 배당률과 같은 의미를 갖는다. 이때, 이자율은 그 자체로 연간 비율로서 표현되기 때문에 주식옵션의 배당과 같은 변환은 필요하지 않다.

[그림 3]은 이러한 옵션가격결정 예를 [그림 1]의 모델 솔버 표현 방식에 따라 보여주고 있다. [그림 3]에서 블랙숄즈방법 솔버인 BLACK_SCHOLES는

EXERCISE_PRICE, MATURITY, DIVIDEND_RATE의 세 입력포트를 가지며 NPV 출력포트를 갖는다. STOCK_OPTION과 CURRENCY_OPTION은 각각 주식옵션과 통화옵션을 나타내며, 여기서는 이들의 포트 중 BLACK_SCHOLES의 포트와 대응되는 포트만을 표시한다. 보다 자세한 포트에 대한 설명은 모델과 솔버간 상호작용에 집중하기 위하여 생략한다.



[그림 3] 옵션가격결정 예

이러한 옵션가격결정 예에서, 이전 절에서 설명한 의미충돌, 이름충돌, 구조충돌의 세가지 모델링 충돌은 다음과 같이 발생한다. 첫째, 의미충돌은 BLACK_SCHOLES의 DIVIDEND_RATE 포트에 대응하는 STOCK_OPTION과 CURRENCY_OPTION의 포트가 서로 다르기 때문에 발생한다. 즉, STOCK_OPTION의 경우 DIVIDEND 포트가 BLACK_SCHOLES의 DIVIDIEND_RATE 포트에 대응되어야 하나 CURRENCY_OPTION의 경우에는 INTEREST_RATE 포트가 대응되어야 한다. 둘째, 이름충돌은 STOCK_OPTION의 EXERCISE_PRICE 포트와 CURRENCY_OPTION의 STRIKE_PRICE 포트가 이음동의어이기 때문에 발생한다. 이 경우, 두 포트 모두는 BLACK_SCHOLES의 EXERCISE_PRICE와 대응되어야 한다. 셋째, 구조충돌은 STOCK_OPTION의 DIVIDEND 포트의 경우 배당 지급일 및 배당금의 리스트로 구성된 매트릭스 형태를 갖게 되나 이에 대응되는 BLACK_SCHOLES의 DIVIDEND_RATE는 연간 비율로 표현되기 때문에 해당 포트들의 포맷을 적절히 변환해야 하기 때문에 발생한다. 그러나, 그 자체가 비율로 표현된 CURRENCY_OPTION의 INTEREST_RATE는 아무런 변경 없이 그대로 전달될 수 있다.

이러한 STOCK_OPTION과 CURRENCY_OPTION의 두 모델과 BLACK_SCHOLES 솔버간 포트 대응 및 데이터 포맷 변환에 대해, 다음 장에서는 본 논문에서 제안하는 모델, 솔버 통합 프레임웍이 어떻게 이들간 인터페이스 규칙을 관리하여 앞서의 세가지 모델링 충돌을 해결하는지를 설명한다. 또한, 인터페이스 규칙을 기반으로 어떻게 제안 프레임웍이 자율적 솔버 제시 및 지능적 모델 해결 기능을 수행하는 지를 설명한다. 이때, 객체지향 설계 개념을 바탕으로 주요 구성체에 대해 간소화된 형태의 클래스 구조를 제시할 것이다.

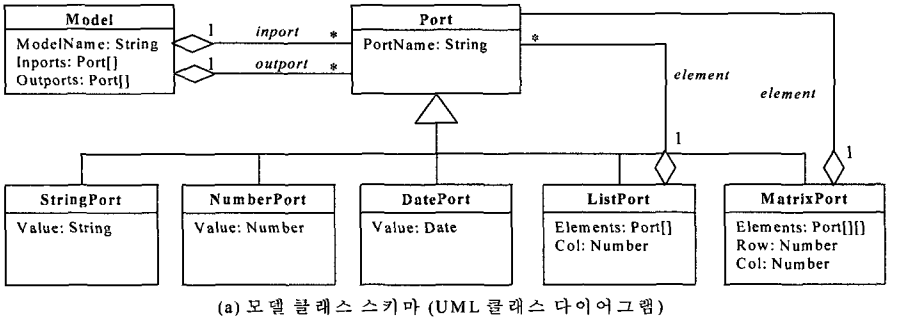
4. 모델, 솔버 통합 프레임웍

이번 장에서는 본 논문에서 제안하는 모델, 솔버 통합 프레임웍의 구체적인 물리적 구조를 제시하고, 이를 구현한 원형시스템에 대해 설명한다. 먼저, 모

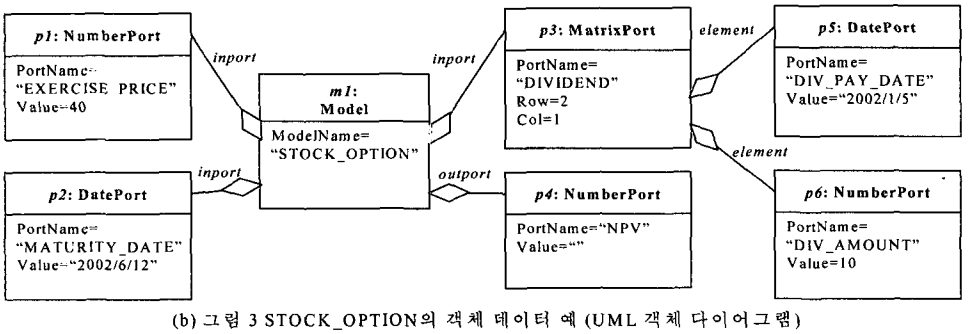
델과 솔버에 대한 데이터 구조인 모델클래스와 솔버클래스를 정의하고, 모델과 솔버간 인터페이스 규칙을 관리하기 위한 정보등록소로서 포트대응사전을 정의한다. 이후, 자율적 솔버 제시 및 지능적 모델 해결 기능을 실행하기 위한 모델에이전트와 솔버에이전트에 대해 설명하며, 마지막으로 원형시스템의 구조 및 실행 화면 예를 제시한다.

4.1 모델클래스

모델클래스는 DSS 내 모델들에 대한 데이터 구조로서, 모델의 외부 인터페이스를 구성하는 포트들을 표현하기 위해 포트클래스를 포함한다. [그림 4](a)는 모델클래스의 객체지향 스키마를 객체지향 모델링 언어인 UML(Unified Modeling Language) 6]을 이용하여 보여준다. [그림 4](a)에서 모델클래스와 포트클래스는 각각 Model과 Port로 명명된다. Model 클래스는 자신의 입력포트와 출력포트를 위해 Port 클래스와 두개의 집단화(aggregation) 관계를 갖는다. Port 클래스는 포트가 가질 수 있는 다양한 형태의 데이터 포맷을 표현하기 위해 NumberPort, StringPort, DatePort, ListPort, MatrixPort 등 몇 개의 하위클래스(subclass)로 다시 세분화된다. 이때, 특정 포트의 데이터 포맷을 결정하기 위해서는 숫자, 문자열, 날짜 등과 같은 데이터 도메인(data domain) 및 스칼라, 리스트, 매트릭스 등과 같은 데이터 차수(data cardinality)가 고려되어야 한다. Port의 하위클래스 중 StringPort, NumerPort, DatePort는 다양한 데이터 도메인을 표현하기 위해 정의된 클래스 이며, ListPort 및 MatrixPort 클래스는 데이터 차수의 표현을 위한 것으로서 Port 클래스와의 element 집단화 관계를 통해 여러 Port 객체를 포함하여 리스트나 매트릭스 값을 나타낸다.



(a) 모델 클래스 스키마 (UML 클래스 다이어그램)



(b) 그림 3 STOCK_OPTION의 객체 데이터 예 (UML 객체 다이어그램)

[그림 4] 모델클래스 정의

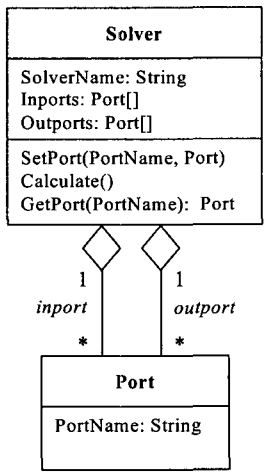
이러한 클래스 다이어그램을 바탕으로 [그림 4] b)는 UML의 객체 다이어그램을 이용하여 [그림 3]의 STOCK_OPTION 모델에 대한 객체 데이터 예를 보여준다. STOCK_OPTION 모델인 m1은 EXE

RCISE_PRICE (p1), MATURITY_DATE (p2), DIVIDEND (p3) 포트를 표현하기 위해 각각 NumberPort, DatePort, MatrixPort 클래스에서 객체화(instantiate)된 세개의 입력포트 객체를 갖는다. 이 중 DIVIDEND 매트릭스(p3)는 배당일(p5)과 배당금(p6)으로 구성된 과거 배당 지급 기록의 집합으로 구성된다. 또한, 해당 모델(m1)은 NPV 포트를 나타내기 위해 NumberPort 객체(p4) 하나를 출력포트로서 포함한다.

4.2 솔버클래스

모델클래스와 같이, 솔버클래스 또한 자신의 포트들을 표현하기 위해 Port 클래스와 inport와 outport의 두가지 일대다 집단화 관계를 가지며, SetPort(), Calculate(), GetPort() 등의 모델 해를 계산하기 위한 연산을 가진다. 즉, 각 연산을 통해 입력포트 값을 지정하고, 해법 알고리즘을 실행하여 해를 계산하며, 그 계산결과를 돌려주는 출력포트의 값을 얻는다. [그림 5](a)는 Solver라 명명된 솔버클래스에 대한 객체지향 스키마를 보여준다.

Solver 클래스는 개별 솔버의 고유 알고리즘을 지원하기 위해 하위클래스들로 다시 특화된다. 특정 솔버를 표현하는 각 하위클래스는 Solver 클래스로부터 모든 속성과 연산을 계승한 후 Calculate() 연산을 커스터마이즈하여 해당 솔버의 특정 알고리즘을 구현한다. 이러한 Solver 클래스와 하위클래스간의 계승 관계는 둘간의 역할과 책임을 효과적으로 분리한다. 즉, Solver 클래스는 Port 클래스와의 집단화 관계를 통한 모델과의 인터페이스 및 DSS와의 통신을 위한 일반적인 외부 구조를 정의하고, 하위클래스는 이렇게 계승된 모델 인터페이스와 DSS 통신 기능을 기반으로 개별 솔버에게 요구되는 고유의 알고리즘을 구현하는 것이다.



(a) 솔버 클래스 스키마 (UML 클래스 다이어그램)

```
/* 솔버 추출 */
define s1 as
  element(select s from s in Solvers
          where s.SolverName = "BLACK_SCHOLES");

/* 입력포트값 설정 */
// p1, p2, p3는 각각 주식옵션모델의 행사가격,
// 만기일, 배당률을 표현함
s1.SetPort("EXERCISE_PRICE", p1);
s1.SetPort("MATURITY_DATE", p2);
s1.SetPort("DIVIDEND_RATE", p3);

/* 계산 실행 */
s1.Calculate();

/* 결과에 대한 출력포트값 얻음 */
define npv as s1.GetPort("NPV");
```

(b) 솔버 사용을 위한 명령 구문의 예

[그림 5] 솔버 클래스 정의

[그림 5](b)는 [그림 3]의 BLACK_SCHOLES 솔버 예를 바탕으로 Solver 객체들을 어떻게 다루는지를 보여주기 위해, OQL(Object Query Language)[11] 언어를 사용한 객체 조작 명령의 예를 보여준다. OQL 언어는 Object Data Management Group (ODMG) 표준[10]에 명시된 객체 조회를 위한 언어로서 SQL과 유사한 형태를 갖는다. [그림 5](b)의 명령구문은 다음과 같은 네 부분으로 구성된다. 첫 부분은 기업 내 모든 솔버 객체의 집합인 솔버 라이브러리로부터 특정 솔버(BLACK_SCHOLES 솔

버)를 추출하고 이를 로컬 객체 s1으로 정의하는 과정이다. 이때, s1은 Solver 클래스의 객체이며 동시에 Calculate() 연산에 블랙숄즈방법 알고리즘을 구현한 하위클래스의 객체이다. 둘째 부분은 어떻게 솔버의 입력포트를 지정하는지를 보여주는 구문으로, 행사가격, 만기일, 배당률을 나타내는 세개의 Port 객체, p1, p2, p3를 s1의 SetPort() 연산을 통해 입력포트로 지정하고 있다. 셋째 부분은 Calculate() 연산을 호출하여 NPV를 계산하는 과정을, 넷째 부분은 s1의 NPV 출력포트로부터 계산 결과를 얻어 이를 npv로 정의하는 과정을 보여준다.

[그림 5](b)의 명령구문에서 본 바와 같이, Solver 클래스와 그 하위클래스간 계승 메커니즘은 특정 솔버 객체인 s1을 해당 솔버에 대한 추가적인 연산이나 데이터 값이 없이도 Solver 클래스의 SetPort(), Calculate(), GetPort()의 세가지 연산을 사용하여 다룰 수 있도록 한다. 이는 DSS가 솔버 라이브러리 내 개별 Solver 객체들을 그들의 고유한 알고리즘 구현방식에 관계없이 단일하게 관리할 수 있도록 한다.

4.3 포트대응사전

포트대응사전(port mapping dictionary)은 모델과 솔버간 포트대응 및 데이터 포맷 변환 방법에 대한 인터페이스 규칙을 관리하기 위한 도구로서, 이들 규칙을 테이블 형태로 관리한다. 즉, 포트대응사전 내 각 행은 대응되어야 할 특정 포트 조합간 인터페이스 규칙을 나타내며, 개별 행은 모델, 솔버, 모델포트, 솔버포트, 포트타입, 변환 스크립트의 여섯 가지 속성(열)으로 구성된다. 모델과 솔버 속성은 해당 인터페이스 규칙이 적용될 모델과 솔버의 이름을 나타내며, 모델포트와 솔버포트는 연결될 모델과 솔버의 해당 포트이름을 표시한다. 포트타입 속성은 해당 대응이 입력포트에 대한 것인지 출력포트에 대한 것인지를 나타내고, 마지막으로 변환 스크립트는 데이터 포맷 변환에 대한 구체적 지침을 서술한다.

이중 변환 스크립트는 포트참조(port reference)와 환경변수(environment variable)의 두 가지 피연산자로 구성된 수식으로서 표현된다. 포트참조는 특정 포트 값을 나타내며 양 옆에 '&'기호를 붙인 포트이름으로 표기한다(예 : &NPV&). 환경변수는 사전에 정의된 시스템 변수이며 대문자열로 표기한다(예 : 현재날짜를 나타내는 TODAY). 이러한 피연산자는 여러 연산자들을 통해 반복적으로 연결되어 변환 스크립트를 구성한다. 스크립트의 연산자로는 데이터 도메인 변환 연산자(예 : 숫자로 변환하기 위한 TO_NUM 연산자, 문자열로 변환하기 위한 TO_STR 연산자 등), 데이터 차수 변환 연산자(예 : 리스트의 합산을 위한 SUM 연산자, 평균을 위한 AVG 연산자 등), 그리고 그 외의 다양한 산술계산을 위한 수학 연산자(예 : 로그 계산을 위한 LOG 연산자, 제곱근 계산을 위한 SQRT 연산자 등) 등이 있다. 변환 스크립트를 위한 모든 가능한 피연산자와 연산자의 종류를 다루는 것은 본 논문의 범위를 벗어나므로 여기서는 스크립트의 기본적 요구사항과 실제적 사용에만 중점을 두기로 한다.

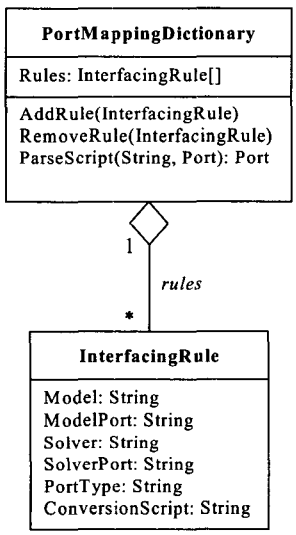
[그림 6]은 [그림 3]의 STOCK_OPTION 모델, CURRENCY_OPTION 모델, BLACK_SCHOLES 솔버의 예를 이용하여 포트대응사전의 개념적 구조를 보여준다. 이중 STOCK_OPTION 모델의 DIVIDEND 포트와 BLACK_SCHOLES 솔버의 DIVIDEND_RATE 포트간 변환 스크립트는 다음과 같다(세번째 행).

```
&DIVIDEND_RATE& = LOG(1 + SUM
    (&DIVIDEND&[*][2]) / &STOCK_PRICE&)
```

위의 수식은 배당일과 배당금의 두 열로 구성된 DIVIDEND 매트릭스를 BLACK_SCHOLES의 DIVIDEND_RA TE 포트가 요구하는 연간 비율로 변환하는 방법을 보여준다. 이때, DIVIDEND 매트릭스의 각 원소를 지정하기 위한 행 인덱스와 열 인덱스로서 두 쌍의 대괄호를 사용한다. 즉, SUM(&DIVIDEND&[*][2])는 DIVIDEND 매트릭스 둘째 열의 모든 원소를 더하는 것을 표현하고 있으며 이는 배당금의 총합이다. 변환 스크립트에 대한 보다 세밀하고 공식적인 설계는 다른 연구를 통해 진행중에 있다[26].

모델	솔버	모델포트	솔버포트	포트타입	변환 스크립트
STOCK_ OPTION	BLACK_ SCHOLES	EXERCISE_ PRICE	EXERCISE_ PRICE	입력	
STOCK_ OPTION	BLACK_ SCHOLES	MATURITY_ DATE	MATURITY	입력	&MATURITY& = (&MATURITY_DATE& - TODAY) / 365
STOCK_ OPTION	BLACK_ SCHOLES	DIVIDEND	DIVIDEND_ RATE	입력	&DIVIDEND_RATE& = LOG(1+SUM(&DIVIDEND&[*][2]) / &STOCK_PRICE&)
STOCK_ OPTION	BLACK_ SCHOLES	NPV	NPV	출력	
CURRENCY_ OPTION	BLACK_ SCHOLES	STRIKE_ PRICE	EXERCISE_ PRICE	입력	
CURRENCY_ OPTION	BLACK_ SCHOLES	MATURITY_ DATE	MATURITY	입력	&MATURITY& = (&MATURITY_DATE& - TODAY) / 365
CURRENCY_ OPTION	BLACK_ SCHOLES	INTEREST_ RATE	DIVIDEND_ RATE	입력	
CURRENCY_ OPTION	BLACK_ SCHOLES	NPV	NPV	출력	

[그림 6] 포트대응사전의 개념적 구조



(a) 포트대응사전 클래스 스키마 (UML 클래스 다이어그램)

```
/* 특정 모델의 적용가능 솔버 조회 */
select distinct r.Solver
from r in PortMappingDictionary.rules
where r.Model = "STOCK_OPTION";


/* 인터페이스 규칙 조회 */
select r.ModelPort, r.SolverPort,
       r.ConversionScript
from r in PortMappingDictionary.rules
where r.Model = "STOCK_OPTION"
  and r.Solver = "BLACK_SCHOLES"
  and r.PortType = "Inport"


/* 포트의 데이터 포맷 변환 */
// script1은 STOCK_OPTION의 DIVIDIEND 포트
// 의 데이터 포맷 변환을 위한 변환 스크립트
define sp
    as PortMappingDictionary.ParseScript
      (script1,
       element(select mp
         from m in Models, mp in m.inports
         where mp.PortName = "DIVIDEND"
         and m.ModelName = "STOCK_OPTION"))
```

(b) 포트대응사전의 사용을 위한 명령구문의 예

[그림 7] 포트대응사전 클래스 정의

[그림 7](a)는 PortMappingDictionary로 명명된 포트대응사전 클래스에 대한 객체지향 스키마를 보여준다. [그림 7](a)에서 PortMappingDictionary 클래스는 인터페이스 규칙을 포함하기 위해 인터페이스 규칙 클래스(InterfaceRule)를 일대다 집단화 관계(rules)로 포함한다. 또한, DSS가 인터페이스 규

칙을 조작할 수 있도록 지원하기 위해 AddRule(), RemoveRule(), ParseScript()의 세가지 연산을 제공한다. AddRule()과 RemoveRule()은 인터페이스 규칙의 등록과 삭제를 위한 것이며, ParseScript()는 변환 스크립트의 해석과 실행을 위한 것이다.

[그림 7](b)는 DSS가 자율적 솔버 제시와 지능적 모델 해결을 수행하기 위해 포트대응사전을 어떻게 이용할 수 있는지에 대한 명령구문의 예를 보여준다. 첫 구문은 포트대응사전으로부터 특정 모델(STOCK_OPTION 모델)의 적용가능 솔버명을 추출하는 방법을 보여준다. 한 모델의 적용가능 솔버는 포트대응사전 내에 해당 모델과의 인터페이스 규칙을 가지고 있을 것이기 때문에, 해당 모델(STOCK_OPTION)과 연관된 인터페이스 규칙들을 찾아내고 이로부터 적용가능 솔버들의 목록을 구할 수 있다. 이와 같은 방법으로, DSS는 포트대응사전을 참조하는 것만으로 사용자의 직접적 개입 없이 적용가능 솔버를 자율적으로 파악하고 제시할 수 있다. 둘째 구문은 포트대응사전으로부터 STOCK_OPTION 모델과 BLACK_SCHOLES 솔버간 입력포트에 대한 인터페이스 규칙을 추출하는 방법을 보여준다. 이 구문을 통해 얻어진 인터페이스 규칙을 이용하여 이들의 포트를 적절히 대응시킬 수 있으며, 데이터 포맷이 다를 경우 이를 적절히 변환할 수 있다. 셋째 구문은 포트대응사전으로부터 얻은 변환 스크립트인 script1에 따라 DIVIDEND 포트의 데이터 포맷을 변환하는 방법이다. 이 구문을 통해 얻어지는 변환된 포트(sp)는 추후 대응되는 솔버의 포트(예 : BLACK_SCHOLES의 DIVIDEND_RATE)에 지정될 것이다.

4.4 모델에이전트와 솔버에이전트

이번 절에서는 Model, Solver, PortMappingDictionary 클래스를 기반으로, 자율적 솔버 제시 및 지능적 모델 해결 기능을 수행하는 두 에이전트모델에이전트(model agent)와 솔버에이전트(solver agent)를 정의한다.

먼저, 사용자 뷰(user view)와 직접 상호작용하는 모델에이전트의 역할은 사용자에게 풀고자 하는 모델의 적용 가능 솔버를 제공하고, 해당 모델의 해가 구해지면 이를 다시 사용자에게 알려주는 것이다. 적용가능 솔버를 제공하기 위해 모델에이전트는 [그림 7](b)에서 제시한 구문과 같은 명령을 이용하여 포트대응사전에서 적용가능 솔버를 찾는다. 이렇게 얻어진 솔버명은 사용자 뷰에 제시되고, 사용자는 제시된 솔버 중 자신의 문제 해결 목적에 따라 적절한 솔버를 선택할 것이다. 사용자가 솔버를 선택하고 모델 해결을 요청하면 모델 에이전트는 모델 해결 요청 메시지를 솔버에이전트에게 보내며, 솔버에이전트가 모델의 해를 반환하면 모델에이전트는 사용자 뷰를 통해 이를 사용자에게 제공한다. 이때, 모델 해결 요청 메시지를 만드는데 있어, 모델에이전트는 모델의 포트로부터 솔버의 입력포트 값을 어떻게 생성하는지 이해하기 위해 포트대응사전의 모델과 솔버간 인터페이스 규칙을 참조할 것이다.

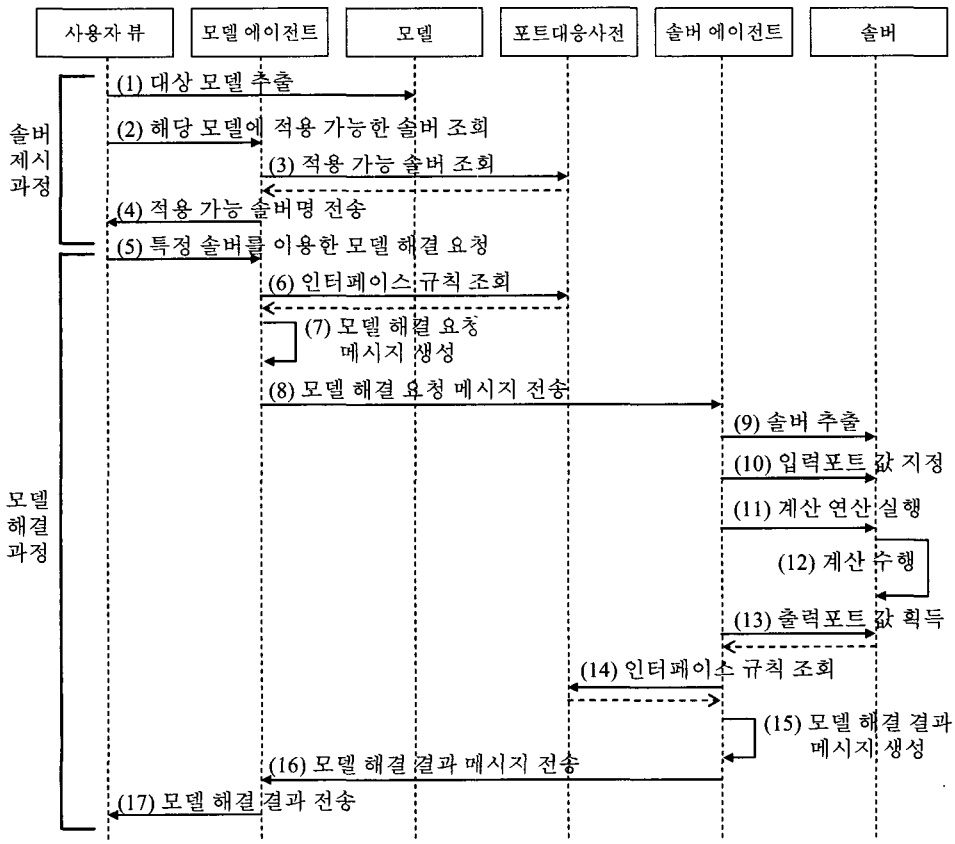
한편, 솔버에이전트의 역할은 모델에이전트로부터 전해지는 모델 해결 요청 메시지에 대해, 모델의 해를 구하고 이를 다시 모델에이전트에게 반환하는 것이다. 모델에이전트로부터 모델 해결 요청 메시지를 받으면, 솔버에이전트는 솔버 라이브러리로부터 메시지에 명시된 솔버를 추출하고 메시지 내의 입력포트 값을 해당 솔버의 입력포트에 지정한 후 모델의 해를 구하기 위한 계산 연산을 실행한다. 이러한 솔버 실행 과정은 [그림 5](b)의 명령과 같은 구문을 통해 수행된다. 솔버의 계산이 끝나면, 솔버에이전트는 모델 해결 결과 메시지를 모델 에이전트에게 보낸다. 이때, 해당 결과 메시지를 생성하기 위해 솔버에이전트는 포트대응사전으로부터 모델과 솔버간 출력포트 대응에 대한 인터페이스 규칙을 참조한다.

[그림 8]은 모델에이전트와 솔버에이전트에 의해 수행되는 이러한 솔버 제시 및 모델 해결 과정의 구체적 단계를 UML 시퀀스 다이어그램을 이용하여 보여준다. 먼저, 솔버 제시 과정은 다음과

같은 4단계로 구성된다. (1) 사용자가 특정 모델의 해를 구하고자 하면, 사용자 뷰는 모델베이스로부터 해당 모델을 추출하고, (2) 이 모델에 대해 적용 가능한 솔버를 모델에이전트에게 조회한다. (3) 모델에이전트는 적용 가능 솔버의 이름을 찾기 위해 포트대응사전을 조회하여 (4) 조회된 솔버 이름을 사용자 뷰에 전달함으로써 솔버 제시 과정이 마무리된다. 사용자는 사용자 뷰에 제시된 솔버명 중 자신의 문제해결목적에 따라 적절한 것을 선택할 것이다.

솔버 제시 과정 이후, 모델 해결 과정은 다음과 같은 단계로 수행된다. (5) 사용자가 특정 솔버를 선택하면, 사용자 뷰는 모델에이전트에게 사용자가 선택한 솔버명과 함께 모델 해결 요청을 보낸다. (6) 모델에이전트는 해당 모델과 솔버간 인터페이스 규칙을 포트대응사전으로부터 조회하여 (7) 솔버의 입력포트 값을 생성하고 (8) 이를 해당 솔버의 이름과 함께 모델 해결 요청 메시지로서 솔버에이전트에게 보낸다. (9) 솔버 에이전트는 솔버 라이브러리로부터 메시지에 명시된 솔버를 추출하고, (10) 해당 솔버의 입력포트 값을 설정한 후 (11) 계산 연산을 실행한다. (12) 계산이 종료되면, (13) 솔버의 출력포트로부터 계산 결과를 얻어 (14) 포트대응사전으로부터 조회한 출력포트대응에 대한 인터페이스 규칙에 따라 (15) 모델 해결 결과 메시지를 생성하고, (16) 이를 모델 에이전트에게 보낸다. (17) 메시지를 받은 모델 에이전트는 사용자 뷰를 통해 모델의 해를 사용자에게 제공한다.

[그림 8]의 과정에서 보듯이, 모델에이전트와 솔버에이전트는 서로 통신하며 포트대응사전에서 관



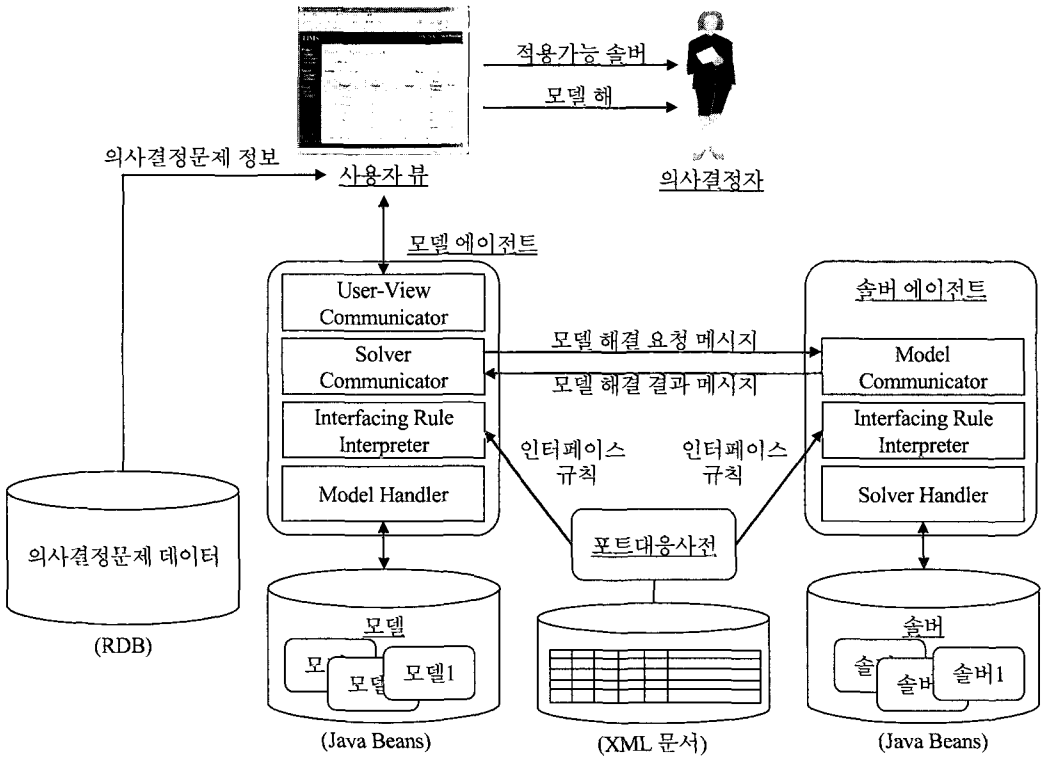
[그림 8] 솔버 제시 및 모델 해결 과정 (UML 시퀀스 다이어그램)

리되는 인터페이스 규칙을 참조하여, 솔버 제시 및 모델 해결 기능을 자율적으로 수행한다. 인터페이스 규칙은 개별 모델과 적용 가능 솔버간 포트대응 및 데이터 포맷 변환 방법을 명시하고 있으므로, 두 에이전트는 앞서 언급한 세가지 모델링 충돌을 지능적으로 해결하며 모델과 솔버간 포트 값을 주고받는다. 이러한 두 에이전트의 자율성 및 지능성은 사용자가 제시되는 솔버 중 적절한 것을 선택하는 것만으로 모델의 해를 쉽게 구할 수 있도록 한다. 또한, 모델의 해결과정을 단순화하고 높은 사용자 편의성을 제공함으로써 비전문 보통 사용자도 DSS가 제공하는 다양한 모델과 솔버를 쉽게 이용할 수 있도록 하여 DSS의 생산성과 유용성을 향상시킬 것이다.

4.5 원형시스템 구현

이번 절에서는 본 논문에서 제안하는 모델, 솔버 통합 프레임웍을 구현한 원형시스템을 통해 제안 프레임웍의 실제 구현 모습을 제시하고 자율적 솔버 제시 및 지능적 모델 해결 기능의 실행 가능성을 논의한다.

[그림 9]는 원형시스템의 구조를 보여준다. 제안 프레임웍의 주요 구성체인 모델, 솔버, 포트대응사전, 그리고 모델 및 솔버 에이전트는 모두 자바프로그래밍언어로 작성된 객체로서 구현되었으며, 이중 포트대응사전은 각 모델과 솔버간 인터페이스 규칙에 대한 정보를 XML 문서로써 저장, 관리한다. 그리고, 시스템의 사용자 뷰는 해결하려는 의사결정 문제에 대한 데이터를 읽어 들이고 이들 주요 구성 객체들과 상호작용하여 문제의 해를 구하고 이를 사용자에게 제공한다. 자율적 솔버 제시 및 지능적 모델 해결 기능을 수행하기 위해 모델 에이전트와 솔버 에이전트는 상호간 데이터 교환을 위한 통신



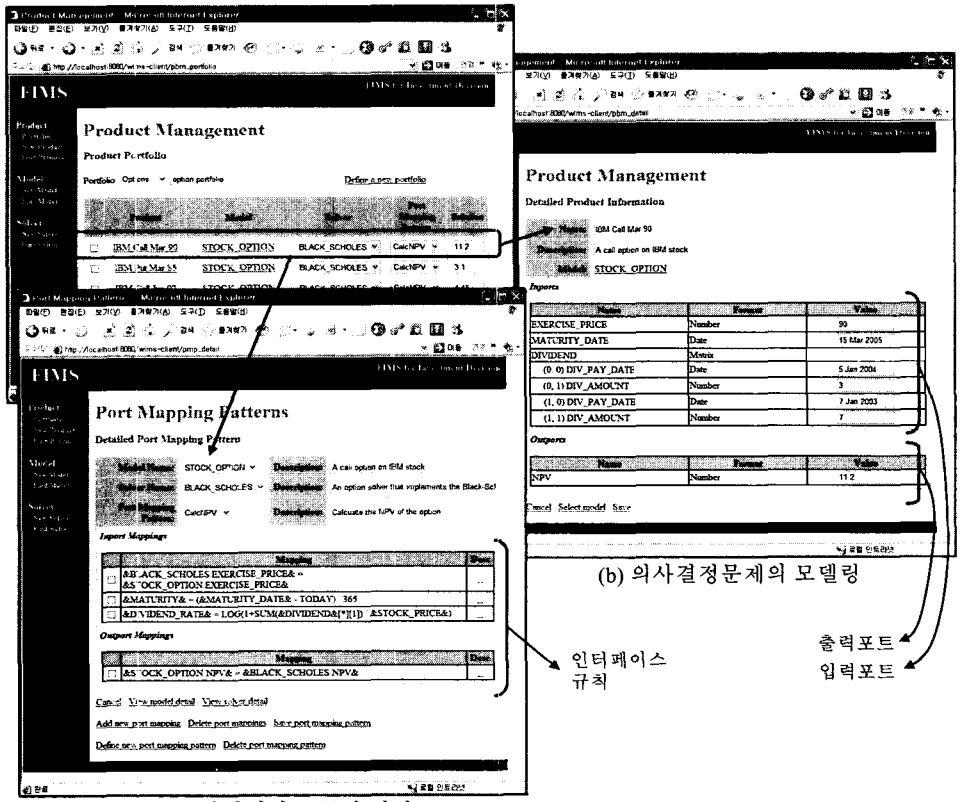
[그림 9] 원형시스템 구조

모듈(model/solver communicator), 포트대응사전이 제공하는 인터페이스 규칙을 읽고 이해하기 위한 해석 모듈(interfacing rule interpreter), 그리고 개별 모델 객체 및 솔버 객체를 다루기 위한 제어 모듈(model/solver handler) 등의 세부 모듈을 갖는다. 또한, 모델 에이전트는 추가로 사용자 뷰와 상호 작용하기 위한 통신 모듈(user-view communicator)을 갖는다.

[그림 10]은 [그림 3]에서 설명한 옵션가격결정 예를 통하여 원형시스템의 화면 예를 보여준다. [그림 10](a) 화면은 옵션 상품(의사결정문제)으로 구성된 포트폴리오의 현재가치를 계산하는 것으로서 각 상품은 해당 상품에 대한 모델을 통해 구조화된다. [그림 10](b)는 'IBM Call Mar 90'이라는 옵션 상품을 STOCK_OPTION 모델을 통해 모델링한 내용을 보여주는 화면으로, STOCK_OPTION 모델의 각 포트에 대해 해당 상품의 데이터 값이 지정되어 있다. 이와 같은 각 상품의 기반 모델에 맞춰 시스템은 해당 상품의 현재가치를 구하기 위해 적용 가능한 솔버 목록을 제시하여 사용자가 적절한 것을 선택할 수 있도록 한다(그림 10(a)). 사용자는 제시된 솔버 중 원하는 것을 선택하고 상품의 현재가치 계산을 요청(모델의 해결을 요청)함으로써 각 상품의 현재가치를 얻을 수 있다.

이러한 솔버 제시 및 모델 해결 과정은 해당 모델과 솔버간 정의된 인터페이스 규칙에 따라 수행되는데, [그림 10](c) 화면은 STOCK_OPTION 모델과 BLACK_SCHOLES 솔버간 인터페이스 규칙을 정의하는 화면을 보여준다. [그림 10](c)에서는 세개의 입력포트 대응과 한 개의 출력포트 대응에

(a) 의사결정문제 해결



(b) 의사결정문제의 모델링

(c) 인터페이스 규칙 정의

[그림 10] 원형시스템의 의사결정문제 해결 예

대한 인터페이스 규칙을 정의하고 있는데, 그 내용은 [그림 6]의 포트대응사전 예에서와 동일하다. 이러한 인터페이스 규칙은 문제 해결 이론에 대해 전문적 지식을 가진 모델/솔버 제작자 혹은 고급 사용자가 정의하며, 전문 지식이 부족한 보통의 일반 사용자는 이와 같이 사전에 정의된 인터페이스 규칙을 활용하여 문제를 해결하게 된다. 즉, 일반 사용자는 단지 적절한 솔버를 선택하는 것만으로 의사결정문제를 해결할 수 있으며, 이는 DSS의 사용자 편의성을 높이고 의사결정문제의 해결에 DSS가 보다 넓게 이용되는데 기여할 것이다.

5. 제안 프레임웍의 의의 및 한계

이번 장에서는 제안하는 모델, 솔버 통합 프레임웍의 의의와 한계 및 그에 따른 향후 연구 방향에 대해 논의한다. 먼저 제안 프레임웍의 의의는 다음과 같이 세가지로 정리될 수 있다.

첫째, 본 논문에서는 모델, 솔버 통합 프레임웍을 금융 의사결정의 범주에서 개발하였으나, 개발 방법론 자체는 모델 기반 DSS가 적용될 수 있는 광범위한 도메인에 대해 적용 가능하다. 이러한 도메인 일반성은 제안 프레임웍의 주요 구성체인 모델, 솔버, 포트대응사전, 모델에이전트, 솔버에이전트 등이 특정 모델링 패러다임에 종속되지 않는 형태로 정의된 것에 기인한다. 특히, 모든 개별 모델 구성 문장 및 솔버 파라미터는 그의 데이터 포맷에 관계없이 포트로서 단일하게 정의되었으며, 따라서 모델과 솔버간 모든 상호작용은 모델포트와 솔버포트간 데이터 교환으로 간주되었다. 이러한 표준화된 모델과 솔버간 인터페이스 구조를 기반으로, 포트대응사전과 모델에이전트 및 솔버에이전트는 솔버 제시와 모델 해결 기능의 모든 기본적인 메커니즘을 다양한 모델과 솔버에 대해서도 일반적으로 적용 가능한 형태로 수행한다. 이와 같은 도메인 일반성은 제안 프레임웍의 주요 구성체를 시스템의 기반 구조물(building block)로서 사용하여 다양한 분야에서 자율적 솔버 제시 및 지능적 모델 해결 능력을 갖춘 DSS를 쉽게 개발할 수 있는 바탕이 될 것이다.

둘째, 모델과 솔버간 인터페이스 규칙이 DSS 내에 하드코딩 되지 않고 포트대응사전에서 독립적으로 관리됨에 따라, 제안 프레임웍은 모델과 솔버의 변화에 대해 쉽게 적응할 수 있다. 즉, 새로운 모델 혹은 솔버가 추가되거나 기존의 것이 변경될 경우, 전체 시스템의 재개발이나 재컴파일 없이 포트대응사전 내의 관련 인터페이스 규칙만을 갱신함으로써 모델 및 솔버의 변화를 쉽게 반영하여 기존의 개체들과 바로 동작할 수 있도록 한다. 이러한 제안 프레임웍의 변화에 대한 적응성은 경영환경의 변화 및 문제 해결 이론의 발전에 따라 모델과 솔버가 지속적으로 변경되는 상황에서 DSS의 유지보수 비용을 크게 줄일 수 있다.

셋째, 자율성, 지능성, 협력성을 갖춘 모델에이전트와 솔버에이전트를 통해 솔버 제시 및 모델 해결 기능을 수행함으로써, 사용자의 모델 해결 과정을 능률화하고 편의성을 확보한다. 모델에이전트와 솔버에이전트는 자율적으로 상호 협력하면서 포트대응사전에서 제공되는 인터페이스 규칙을 지능적으로 이해하여, 자율적 솔버 제시 및 지능적 모델 해결 기능을 수행한다. 따라서, 사용자는 모델 해결 과정에 대한 자세한 지식 없이도 다양한 솔버를 이용하여 모델을 풀어볼 수 있다. 나아가 모델에이전트와 솔버에이전트는 기업 내외부 네트워크 상에 모델과 솔버가 분산되어 구축된 분산 DSS 환경에서 분산된 모델과 솔버를 확인하고 통합하는데 있어 더욱 중요한 역할을 수행할 수 있다. 예를 들어, 모델과 솔버의 네트워크 상의 위치, 시스템 플랫폼 등에 관계없이 특정 의사결정 문제에 적용 가능한 모델과 솔버 조합을 발견하고 실행할 수 있다. 이를 위해, 분산 환경에서 모델에이전트와 솔버에이전트의 역할을 확대하고 웹서비스 기술(Web Services) [7]을 접목한 확장 모델, 솔버 통합 프레임웍에 대한 추가적인 연구를 수행하고 있다[25].

이상의 연구의의와 함께 제안 프레임웍의 한계는 다음과 같이 세가지로 정리될 수 있다. 각각의 한계

와 함께 이를 보완하기 위한 향후 연구방향을 제시한다.

첫째, 제안 프레임웍은 사용자가 풀고자 하는 모델에 대해 적용 가능한 솔버 집합을 제시함으로써 사용자의 솔버 선택을 도우나, 결국 제시된 솔버 중 최종적으로 이용할 솔버를 사용자가 직접 선택해야 한다는 한계가 존재한다. 이러한 사용자의 최종 솔버 선택에 대한 제약은 제안된 솔버들의 의미를 이해하고 서로 비교할 수 있는 능력을 요구하므로 DSS 사용자의 범위를 경험 많은 사용자나 분석가로 제한하는 원인이 될 수 있다. 이에 대해, 솔버 제작자는 솔버의 목적, 기본 가정, 계산 결과의 의미 등을 명시한 문서를 제공함으로써 사용자가 제공 문서를 참조하여 적절한 솔버를 선택할 수 있도록 할 수 있을 것이다. 나아가, RDF(Resource Description Framework)[28] 기술과 같은 XML 기반의 시스템 인식가능 메타데이터 관리 기술이 발전함에 따라 솔버에 대한 이러한 문서를 시스템이 이해할 수 있는 형태로 제작함으로써 솔버 선택 과정을 완전 자동화할 수 있을 것이다.

둘째, 포트대응사전의 모델과 솔버간 인터페이스 규칙은 모든 적용 가능한 모델과 솔버의 조합에 대해 명시되어야 하며 이의 설정을 담당하는 모델링 전문가에 의해 수동으로 관리된다. 이와 같은 인터페이스 규칙의 수동관리는 모델과 솔버의 수가 늘어남에 따라 심각한 유지보수 문제를 야기할 수 있다. 이러한 인터페이스 규칙 관리의 부담을 줄이기 위해 향후 연구 방향으로서 솔버를 공유할 수 있는 유사 모델끼리 그룹화(즉, 모델 타입[20] 혹은 모델 클래스[8])한 모델 분류 체계의 설계를 고려하고 있다[25]. 모델 분류 체계가 확립되면 개별 모델이 아닌 모델 그룹에 대해 모델과 솔버간 인터페이스 규칙을 정의할 수 있으므로 관리되어야 할 규칙의 수를 크게 줄일 수 있어 시스템 유지보수의 문제를 상당부분 해소할 수 있을 것이다.

셋째, 모델링 전문가는 모델과 솔버간 인터페이스 규칙 내에 포트간 데이터 포맷을 변환하는 변환 스크립트를 작성함으로써 모델포트와 솔버포트간 포맷의 차이를 조절한다. 본 논문에서는, 모델과 솔버의 통합시 발생 가능한 모델링 충돌 및 이의 해결에 바람직한 시스템 아키텍처에 중점을 두어 이러한 변환 스크립트의 기본 요구사항 및 사용방법에 대해 논의하였다(보다 자세한 논의는 참고문헌 [3, 8] 참조). 그러나, 실제의 경우 변환 스크립트의 관리는 훨씬 복잡해질 수 있으며 이는 시스템의 유지보수에 있어 문제가 될 수 있다. 이에 대해, 향후 연구 과제로서 보다 유연하고 외부 환경 변화에 대해 효과적으로 적응할 수 있는 변환 스크립트 구조를 설계하고 있다[26]. 또한, 비전문 사용자도 자신의 문제 해결 목적을 반영하여 스크립트를 쉽게 작성할 수 있는 사용자 인터페이스도 개발 중이다.

6. 결 론

DSS를 이용하여 의사결정문제를 해결하고자 하는 사용자에게 있어, DSS가 제공하는 모델과 솔버의 의미 및 구조를 이해하고 그들간 적용가능성 및 파라미터간 대응 패턴을 파악하여 이들을 이용하는 것은 대부분의 경우 매우 어려운 일이다. 따라서, 사용자가 쉽게 솔버를 선택하고 적용할 수 있도록 지원하는 것은 DSS의 중요 역할 중 하나이다. 이를 위해, 본 논문은 DSS가 특정 모델의 적용 가능 솔버를 사용자의 직접적인 개입 없이 자율적으로 제시할 수 있고 심각한 모델링 충돌(의미충돌, 이름충돌, 구조충돌)이 없이 모델과 솔버간 파라미터를 지능적으로 대응시켜 모델을 해결할 수 있도록 하는 모델과 솔버의 통합 프레임웍을 제안하였다. 제안 프레임웍을 구축하는데 있어, 본 논문은 모델과 솔버를 입력포트와 출력포트로 구성된 기반 구조물로서 정의하고, 포트대응과 데이터 포맷 변환 방법을 통해 모델과 솔버간 주요 상호작용을 포괄하는 표준화된 모델 솔버 인터페이스 체계를 정의하였다. 이때, 포트대응과 데이터 포맷 변환 방식은 인터페이스 규칙으로 정의되어 포트대응사전에서 관리되며, 모델에이전트와 솔버에이

전트가 이를 참조하고 서로 통신하며 DSS의 솔버 제시 및 모델 해결 기능을 자율적으로 수행한다. 본 논문의 가장 큰 의의는 모델, 솔버, 포트대응테이블, 모델에이전트, 솔버에이전트 등 제안 프레임웍의 주요 구성체들을 DSS가 적용될 문제 도메인 및 모델링 패러다임에 대한 제약 없이 일반적인 구조를 갖도록 개발함으로써, 모델링 패러다임과 적용 도메인에 독립적이고 자율적 솔버 제시 및 지능적 모델 해결 기능을 갖춘 모델, 솔버 통합 프레임웍을 제안한다는데 있다.

참 고 문 헌

[1] Ba, S., R. Kalakota and A.B. Whinston, "Using Client-Broker-Server Architecture for Intranet Decision Support," *Decision Support Systems*, Vol.19(1997), pp.171-192.

[2] Beynon, M., S. Rasmequan and S. Russ, "A New Paradigm for Computer-Based Decision Support," *Decision Support System*, Vol.33 (2002), pp.127-142.

[3] Bhargava, H.K., S.O. Kimbrough and R. Krishnan, "Unique Names Violations, a Problem for Model Integration or You Say Tomato, I Say Tomahto," *ORSA Journal on Computing*, Vol.3(1991), pp.107-120.

[4] Bhargava, H.K., R. Krishnan, S. Roehrig, M. Casey, D. Kaplan and R. Mller, "Model Management in Electronic Markets for Decision Technologies : A Software Agent Approach," *Proceedings of the 30th Hawaii International Conference on System Sciences*, Vol.5(Jan. 1997), pp.405-415.

[5] Bolloju, N., M. Khalifa and E. Turban, "Integrating Knowledge Management into Enterprise Environments for the Next Generation Decision Support," *Decision Support Systems*, Vol.33(2002), pp.163-176.

[6] Booch, G., J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, Massachusetts, 1999.

[7] Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard, Eds., "Web Services Architecture," *W3C Working Group Note*, (Feb. 2004), http : //www.w3.org/TR/2004/NOTE-ws-arch-20040211.

[8] Bradley, G.H., and R.D. Clemence, Jr., "A Type Calculus for Executable Modeling Languages," *IMA Journal of Mathematics in Management*, Vol.1(1987), pp.277-291.

[9] Brooke, A., D. Kendrick, A. Meeraus and R. Raman, "GAMS : A User's Guide," *GAMS Development Corporation*, (Dec. 1998), http://www.gams.com/docs/gams/GAMSUsersGuide.pdf.

[10] Cattell, R.G.G., D.K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda and F. Velez, Eds., *The Object Data Standard : ODMG 3.0*, Morgan Kaufmann, 2000.

[11] Cluet, S., "Designing OQL : Allowing Objects to Be Queried," *Information Systems*, Vol.23 (1998), pp.279-305.

[12] Dolk, D.R., "Integrated Model Management in the Data Warehouse Era," *European Journal of Operational Research*, Vol.122(2000), pp.199-218.

[13] Dutta, A., "Integrating AI and Optimization for Decision Support : A Survey," *Decision Support Systems*, Vol.18(1996), pp.217-226.

[14] Fourer, R., D.M. Gay and B.W. Kernighan, "A Modeling Language for Mathematical Programming," *Management Science*, Vol.36 (1990), pp.519-554.

[15] Fourer, R., "Database Structures for Mathematical Programming Models," *Decision*

Support Systems, Vol.20(1997), pp.317-344.

[16] Geoffrion, A.M., "An Introduction to Structured Modeling," *Management Science*, Vol. 33(1987), pp.547-588.

[17] Geoffrion, A.M., "The SM Language for Structured Modeling : Levels 1 and 2," *Operations Research*, Vol.40(1992), pp.38-57.

[18] Geoffrion, A.M. and S. Maturana, "Generating Optimization-Based Decision Support Systems," *Proceedings of the 28th Hawaii International Conference on System Sciences*, Vol.3(Jan.1995), pp.439-448.

[19] Gregg, D.G., M. Goul and A. Philippakis, "Distributing Decision Support Systems on the WWW : The Verification of a DSS Metadata Model," *Decision Support Systems*, Vol.32(2002), pp.233-245.

[20] Huh, S., "Modelbase Construction with Object-Oriented Constructs," *Decision Science*, Vol.24(1993), pp.409-434.

[21] Huh, S. and H. Kim, "A Real-Time Synchronization Mechanism for Collaborative Model Management," *Decision Support Systems*, Vol.37(2004), pp.315-330.

[22] Hull, J.C., *Options, Futures, and Other Derivatives*, Prentice Hall, New Jersey, 1997.

[23] Jennings, N.R., K. Sycara and M. Wooldridge, "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems*, Vol.1(1998), pp.275-306.

[24] Kone, M.T., A. Shimazu and T. Nakajima, "The State of the Art in Agent Communication Languages," *Knowledge and Information Systems*, Vol.2(2000), pp.259-284.

[25] Lee, K. and S. Huh, "Model-Solver Integration in Decision Support Systems : A Web Services Approach," *Pre-ICIS SIG-DSS Workshop*, (Dec. 2003).

[26] Lee, K. and S. Huh, "Data Conversion Rules for Integrating Disparate Algebraic Models and Solvers," *Working Paper*, KAIST(2004).

[27] Ma, J., "Type and Inheritance Theory for Model Management," *Decision Support Systems*, Vol.19(1997), pp.53-60.

[28] Manola, F., and E. Miller, Eds., "RDF Primer," *W3C Recommendation*, (Feb. 2004), http : //www.w3.org/TR/2004/REC-rdf-primer-20040210.

[29] Muhanna, W.A. and R.A. Pick, "Meta-Modeling Concepts and Tools for Model Management : A Systems Approach," *Management Science*, Vol.40(1994), pp.1093-1123.

[30] Rizzoli, A.E., J.R. Davis and D.J. Abel, "Model and Data Integration and Re-use in Environmental Decision Support Systems," *Decision Support Systems*, Vol.24(1998), pp. 127-144.

[31] Ryu, Y.U., "Constraint Logic Programming Framework for Integrated Decision Supports," *Decision Support Systems*, Vol.22(1998), pp. 155-170.

[32] Shim, J.P., M. Warkentin, J.F. Courtney, D.J. Power, R. Sharda and C. Carlsson, "Past, Present, and Future of Decision Support Technology," *Decision Support Systems*, Vol.33 (2002), pp.111-126.

[33] Wright, G.P., A.R. Chaturvedi, R.V. Mookerjee and S. Garrod, "Integrated Modeling Environments in Organizations : An Empirical Study," *Information Systems Research*, Vol.9 (1998), pp.64-84.

[34] Zhuge, H., "Inheritance Rules for Flexible Model Retrieval," *Decision Support Systems*, Vol.22(1998), pp.379-390.