Model-Solver Integration in Decision Support Systems: A Web Services Approach

Keun-Woo Lee[a,*], Soon-Young Huh[a]

[a]*Graduate School of Management, Korea Advanced Institute of Science and Technology*

207-43 Chongyang-ni, Dongdaemoon-gu, Seoul, 130-722, Korea

Abstract

As complex mathematical models are increasingly adopted for business decision-making, many decision makers have difficulty in selecting compatible solvers (i.e., model solving algorithms) for a model and adequately applying them to the model. This paper presents a model-solver integration framework that enables a decision support system to autonomously suggest the compatible solvers and to apply them to the model, even though the users are not knowledgeable enough about all the details of the models and the solvers. In developing the framework, this paper adopts a Web services approach for the integration of the models and solvers created on different modeling paradigms by encapsulating individual solvers as Web services.

[*] Corresponding author. Tel.: +82-2-958-3650; fax: +82-2-958-3604.

E-mail addresses: kwlee@kgsm.kaist.ac.kr (K. Lee), syhuh@kgsm.kaist.ac.kr (S. Huh).

1. Introduction

As business environments become more competitive and rapidly change, decision support systems (DSS) for precise and agile decisions have been increasingly adopted in many organizations. In a DSS, for user-friendliness and intuitive solution, a decision problem is formulated as a model and diverse sets of solvers (i.e., model-solving algorithms) are also provided. Particularly, since a model may have multiple problem-solving purposes, it needs multiple solvers depending on the purposes, and conversely, a single solver can be also applied to multiple models that have similar problem structures. Under this many-to-many relationship between models and solvers, to solve a model, a user of the DSS should be knowledgeable about which solvers can be applied to the model. Moreover, once an applicable solver is chosen, the user should also understand how to match the individual model parameters to the solver parameters to execute the solver adequately [5].

In reality, however, since most ordinary DSS users might not be familiar with the entire set of the models and solvers but only specific kinds, they often have difficulty in picking out the applicable solvers of a given model from the organizational solver library. In addition, the complex model-solver interaction semantics, hardly understandable for the ordinary users, make it difficult for them to match the model parameters to the solver parameters in the model-solving process. This is not a trivial task even for the users with specific skill levels in the problem-solving theories, which means that they understand the conceptual objectives and constraints of the solvers and how to set up the initial parameter values of the model to be solved [1,3]. Moreover, as more organizations have constructed the DSS distributed across their internal/external networks [4], models and solvers have been created based on different modeling paradigms and different system platforms. Such heterogeneity of models and solvers makes it more difficult to utilize models and solvers in solving decision problems.

Thus, the DSS is required to support the following two capabilities to make the overall model solution process easy and productive. First, for a specific model under consideration, the DSS should be able to suggest autonomously a set of solvers that are both syntactically and semantically compatible with the model so that the users can select an appropriate one among the suggested solvers. Second, when a particular solver is chosen for the model, the DSS should be able to match the model parameters to the adequate solver parameters intelligently and produce the model solving results even though the user cannot perform exact matching between the two.

Recognizing such requirements of the DSS, this paper focuses on the development of a model-solver integration framework that facilitates the autonomous solver suggestion and intelligent model solution capabilities. In developing the framework, we adopt the Web services technologies [6] as a vehicle for integrating the models and solvers distributed across networks and created based on different modeling paradigms and system platforms. The Web services technologies have been highlighted as a way of integrating distributed and heterogeneous applications, which was previously impractical because of non-interoperable proprietary approaches. In our framework, individual solvers are encapsulated as Web services (called solving services) that provide the model-solving functions. Given these solving services, two intelligent agents, model and solver agents, assist users with solving models by suggesting compatible solving services and matching parameters of the two without direct users' intervention [2]. In addition, for the agents to identify the compatibility of a solving service with a model and their parameter matching patterns, an interfacing ontology managing such interfacing semantics between individual models and solving services is also defined.

2. Model-Solver Integration Framework

Figure 1 shows the conceptual architecture of the model-solver integration framework proposed in this paper. The interfacing ontology and the model and solver agents are to be explained in the following sections.

2.1. Interfacing Ontology

First, for representation of the compatibility between individual models and solving services, models are classified into groups (i.e., model taxonomy) in such a way that every model in a group can share solvers with one another. By assigning solving services to the individual model groups, the interfacing ontology can represent which models a solving service can be applied to. Figure 2 shows example taxonomy of optimization models and assignment of compatible solvers.

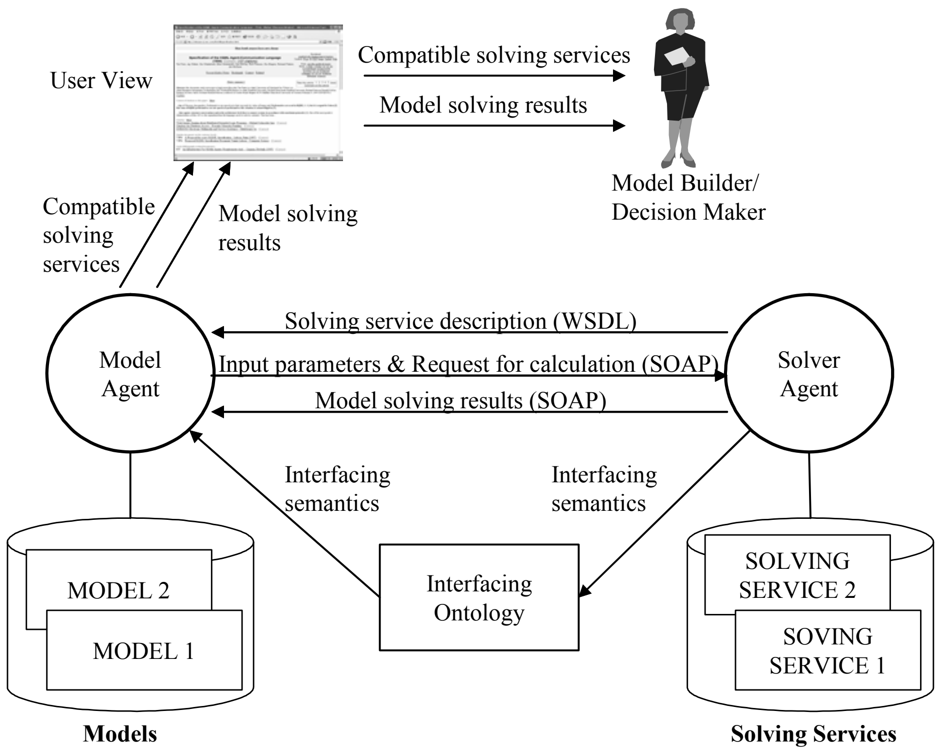

Figure 1. Conceptual Architecture of the Model-Solver Integration Framework.

In model management literature, such model taxonomies have been proposed implicitly or explicitly to organize similar models into groups. However, since those taxonomies are usually concerned only with structural assumptions within the models such as whether a parameter of a model is continuous or discrete, we use more detailed taxonomy where the models are categorized one step further in terms of the shareability of solvers.

Second, for representation of the parameter matching patterns between individual models and their compatible solving services, a parameter mapping dictionary is constructed. The parameter mapping dictionary manages all the possible types of mapping relationships between model parameters and solver parameters in a tabular form where each row represents an individual parameter mapping (Table 1). We can distinguish those parameter mappings according to the following two factors.

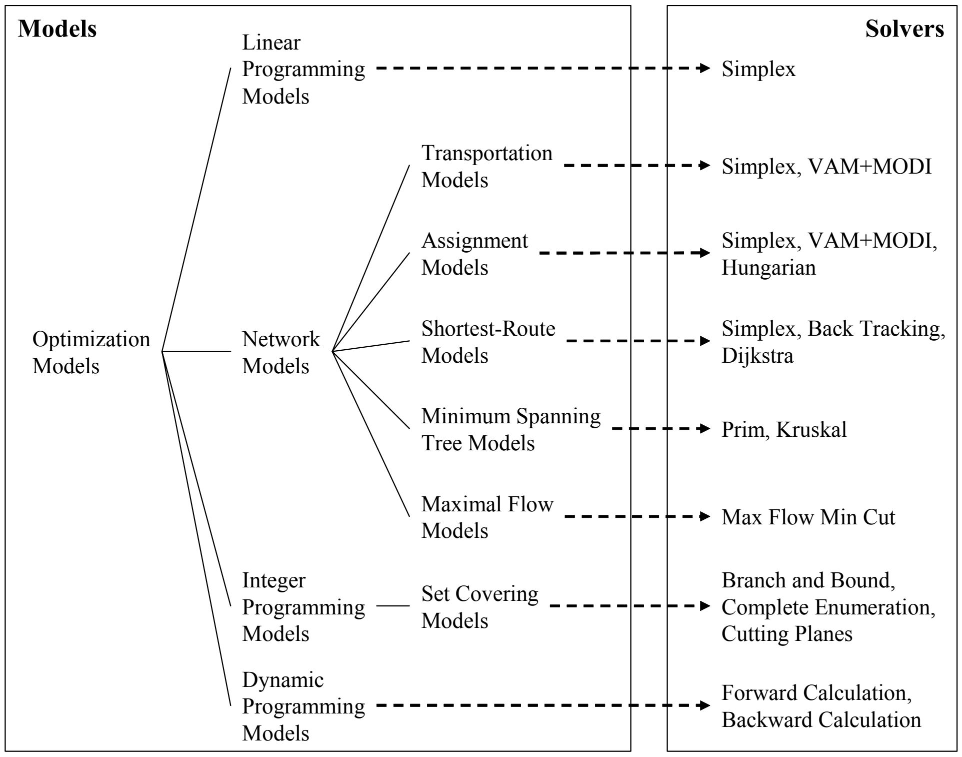
Models
Solvers
Optimization Models
Linear Programming Models
Simplex
Network Models
Transportation Models
Simplex, VAM+MODI
Assignment Models
Simplex, VAM+MODI, Hungarian
Shortest-Route Models
Simplex, Back Tracking, Dijkstra
Minimum Spanning Tree Models
Prim, Kruskal
Maximal Flow Models
Max Flow Min Cut
Integer Programming Models
Set Covering Models
Branch and Bound, Complete Enumeration, Cutting Planes
Dynamic Programming Models
Forward Calculation, Backward Calculation

Figure 2. Example Taxonomy of Optimization Models and Compatible Solver Assignment

- *Cardinality*: A parameter mapping can have 1:1, 1:*n*, *n*:1, or *n*:*m* cardinality. An 1:1 or *n*:1 mapping is specified in a single row of the dictionary; an 1:*n* mapping is converted to *n* 1:1 mappings and thus specified in *n* rows; an *n*:*m* mapping is converted to *m* *n*:1 mappings and thus specified in *m* rows. For example, in Table 1, the first row indicates the 1:1 mapping between PARAM 1 and PARAM 6, and the second row describes the *n*:1 mapping between the three parameters (PARAM 2, PARAM 3, and PARAM 4) and PARAM 7 by averaging the three parameters. The 1:*n* mapping between PARAM 5 and the two parameters (PARAM 8 and PARAM 9), shown in the third and fourth rows, is converted to two 1:1 mappings.
- *Transformation function*: A parameter mapping can include a transformation function. When the transformation function is applied to the parameter value of the sender, it will produce the data value required in the receiver. A transformation function is expressed as an algebraic formula built from the mapping parameters recursively composed by a set of operators. A parameter is denoted by its name enclosed by ampersands (&) at both ends (e.g., &PARAM 1&). The operators include various mathematical operators for the numerical calculations of the parameter values.

Table 1. A Conceptual Structure of the Parameter Mapping Dictionary.

Model	Model Parameters	Solving Service	Solver Parameters	Transformation Function
MODEL 1	PARAM 1	SERVICE 1	PARAM 6	
MODEL 1	PARAM 2, PARAM 3, PARAM 4	SERVICE 1	PARAM 7	&PARAM 7& = AVG(&PARAM 2&, &PARAM 3&, PARAM 4&)
MODEL 1	PARAM 5	SERVICE 1	PARAM 8	&PARAM 8& = &PARAM 5& / 10
MODEL 1	PARAM 5	SERVICE 1	PARAM 9	&PARAM 9& = &PARAM 5& / 100

By referring to the parameter mapping dictionary, the DSS can identify the compatible solving services of a model and their parameter matching patterns. For example, the following SQL command shows how to retrieve the compatible solving service names of the model MODEL 1 from the parameter mapping dictionary. (Though the parameter mapping dictionary is not a database table, we use commands written in SQL to illustrate how to manipulate the dictionary because it has a tabular form.)

```
select distinct SolvingService
from ParameterMappingDictionary
where Model = "MODEL 1";
```

Since the compatible solving services must have parameter mappings with the model in the dictionary, this command finds the solving services by extracting the attribute *solving service* from the mappings that have the model name "MODEL 1" in the attribute *model*.

2.2. Model and Solver Agents

The model agent acts as an intermediary between a user view and the solver agent to support a user in solving a model. Specifically, the model agent brings two types of information to the user view: compatible solving services with the model and the model solving results. First, when a user view references a model, the model agent consults the interfacing ontology and identifies the compatible solving services. Then, via the user view, the model agent suggests them for the user to select one. Second, when the user selects a solving service to be used, the model agent requests for description of the selected solving service from the solver agent. According to the description, the model agent sends a Simple Object Access Protocol (SOAP) [6] message containing the input parameters required in the solving service and the request for execution of model-solving calculation. In creating the message, the model agent refers to the interfacing ontology again to understand the parameter matching patterns between the model and the solving service. Afterwards, the model agent delivers the solution returned by the solver agent to the user view.

In this context, the solver agent has the following two responsibilities in response to the model agent's request: sending the service description of a solving service and executing the model-solving calculation. First, the service description defines the solving service's interface such as the message formats, data types, and transport protocols that should be used in the service. Such service description is written in a standardized description language such as Web Service Description Language (WSDL) [6]. Second, when the solver agent receives the request for model-solving calculation from the model agent, it executes the solving service and returns the results to the model agent in a SOAP message format.

3. Conclusions

A DSS can achieve high productivity and effectiveness in supporting a user's model-solving activities by satisfying the following two functional requirements: autonomous solver suggestion and intelligent model solution. Having recognized these requirements, this paper proposes a model-solver integration framework that facilitates the two requirements. In developing the framework, we use the Web services technologies to integrate the models and the solvers in unified system architecture. Also, we design an interfacing ontology for the model and solver agents to infer the compatibility of a solving service with a certain model and to understand their parameter matching patterns. Referring to the ontology, the agents can suggest the compatible solvers and solve the models without direct intervention of the users. Thus, the proposed model-solver integration framework can be equipped with the autonomous solver suggestion and intelligent model solution capabilities.

Now, we are working on elaborating the interfacing ontology to provide more concrete and formal specifications about the model-solver compatibility and the parameter matching patterns. A simple ontology that has a form of mapping table has been constructed but it is being extended based on the Resource Description Framework (RDF) [6]. Also, a prototype system for the model-solver integration framework is being developed with JAVA programming language.

References

[1] Beynon M, Rasmequan S, Russ S, “A New Paradigm for Computer-Based Decision Support,” *Decision Support System* 2002; 33(2): 127-142.

[2] Bhargava H, Krishnan R, Roehrig S, Casey M, Kaplan D, Müller R, “Model Management in Electronic Markets for Decision Technologies: A Software Agent Approach,” *Proceedings of the 30th Hawaii International Conference on System Sciences* 1997: 405-415.

[3] Dutta A, “Integrating AI and Optimization for Decision Support: A Survey,” *Decision Support Systems* 1996; 18(3-4): 217-226.

[4] Gregg D, Goul M, Philippakis A, “Distributing Decision Support Systems on the WWW: the Verification of a DSS Metadata Model,” *Decision Support Systems* 2002; 32(3): 233-245.

[5] Huh S, “Modelbase Construction with Object-Oriented Constructs,” *Decision Science* 1993; 24(2): 409-434.

[6] World Wide Web Consortium, http://www.w3.org/, 2003.