A Model Context-Based Solution Framework in Decision Support Systems

Keun-Woo Lee and Soon-Young Huh

Graduate School of Management, Korea Advanced Institute of Science and Technology
207-43 Chongyang-ni, Dongdaemoon-gu, Seoul, 130-722, South Korea
Tel: +82-2-958-3650, Fax: +82-2-958-3604, E-mail: {kwlee, syhuh}@kgsm.kaist.ac.kr

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 in the presence of a decision model, the flexible selection and execution of solvers is still unsatisfactory model and thus solution processes are still too complex for ordinary users to apply appropriate solvers to the model. This paper proposes a model context-based solution framework that enables the user to solve decision problems using models and their associated multiple solvers without requiring precise knowledge of the model-solution processes. Specifically, for intuitive model-solution, the framework enables a model management 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.

Keywords:

Model management systems; Model-solver integration; Decision support systems

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 [21,22]. In a DSS, for user-friendliness and intuitive solution, a decision problem is formulated as a model – a structured form composed of a set of declarative modeling language statements [10,13] that are easy to understand and execute. Specifically, a model management system (MMS) is dedicated to managing the entire life cycle of models as a part of the three component modules of a DSS together with a database management system (DBMS) and dialogue management system [21].

In the model management research area, there have been a wide spectrum of studies ranging from the modeling languages for effective modeling of management science/operations research (MS/OR) problems [6,9,10], and the model representation scheme for easier creation, retrieval, and execution of models [10,13,20], to the DSS component integration framework for the reuse of models with different data sets for different problems [7,17,19]. However, there has been little effort focusing on the precise and flexible decision making support by the reuse of solvers [8,13], i.e., problem-solving algorithms against diverse models and data sets. The reuse of solvers brings in two kinds of model-solver integration perspectives. While a solver can be applied to and reused in multiple models that have similar problem structures, a single model may have multiple-problem solving purposes and thus needs multiple solvers. Such a flexible integration framework can make the decision support process more user-friendly and streamlined, and result in a more simplified system architecture that leads easier implementation and maintenance of DSS.

To sufficiently attain these benefits from the flexible model-solver integration, the user should be knowledgeable of what a model and its component statements mean and which solvers are applicable to the model under certain purposes. Once an applicable solver is chosen, the user should be able to understand how to solve the model, flexibly with the individual model component statements and solver parameter values, to best meet the problem-solving purposes [1,13]. In reality, however, since the semantic understanding of a model or a solver is not a trivial task, ordinary users, even sometimes knowledgeable professional users, often have difficulty in picking out the applicable solvers from the organizational solver library and adequately applying them to a given model [2]. Understanding such a burdensome model and solver semantics makes users less active and often poses obstacles in utilizing models in solving decision problems even though they admit the overall usefulness of the models.

Moreover, as more organizations have constructed the DSS distributed across their internal/external networks [12,14], 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 them in solving decision problems. Thus, the MMS as a dedicated tool for managing the models should be able to support the following two capabilities to make the overall model solution process easy and productive.

First, for a specific model under consideration, the MMS should be user-friendly enough to suggest autonomously a set of solvers that are both syntactically and semantically compatible with the model. This autonomous solver sugges-

tion capability will enable the ordinary user who might not have enough expertise to identify compatible solvers, to select appropriate solvers easily and be prevented from misusing incompatible solvers. Second, when a particular solver is chosen for the model, adequate parameter matching is needed for model-solution between the model and solver in such a way that input values in the model are inserted as the corresponding input parameter values of the solver. In addition, output values in the solver are to be transferred as the corresponding output values of the model. Thus, the MMS should be able to perform the parameter matching between the model and solver intelligently and produce the model solution result even though the user has a little knowledge of the meaning of parameter values of a solver and cannot perform exact matching between model parameters and solver parameters.

Recognizing such requirements of the MMS, this research 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, first, we propose systematic and unified representation schemes for models, solvers, and their interactions by employing the generic model concept [13] as a conceptual framework (Section 2). Specifically, modeling conflict issues are explained, which make it difficult to define a standardized interfacing scheme for the integration of models and solvers. Second, on the basis of the model and solver representation schemes, we propose a multi-agent approach [3,16] as physical system architecture of the framework, and explain system procedures used for the autonomous solver suggestion and intelligent model solution (Section 3). Finally, we discuss the results and contributions of this paper followed by future research directions (Section 4).

Generic Model-Solver Representation

This section describes systematic and unified representations of models, solvers, and their interactions with particular focus on the modeling conflicts among the models. For the illustration of the representations and conflicts, an option pricing example of a financial MMS is provided.

Core Concepts of Model-Solver Representation

We adopt the generic model concept [13] as a conceptual framework to represent models, solvers, and their interactions, and extend it to be more flexible in selecting and applying solvers to a model.

In our representations, a *model*, characterized by an object type model type, concerns the semantic issues of the decision problems, and thus is responsible for maintaining and updating all the problem statements. The model type has a name, a set of ports, and primitive manipulating operations. A *port*, characterized by a port type, corresponds in a one-to-one mapping relationship to a problem statement of a model. The ports, as a whole, constitute the external interface of a model, through which the model interacts with its outside environment (i.e., human modelers, solvers and databases). With respect to the interaction role, the ports of a model are classified into two types: input ports (inports) admit data and output ports (outports) produce problem-solving results in connection with their environment. The attributes of the port type are intended to directly capture the meaning and structure of a single problem statement of the model. They are a unique name, a set of attributes and operations to describe the information pertaining to data values and algebraic expression of the statement.

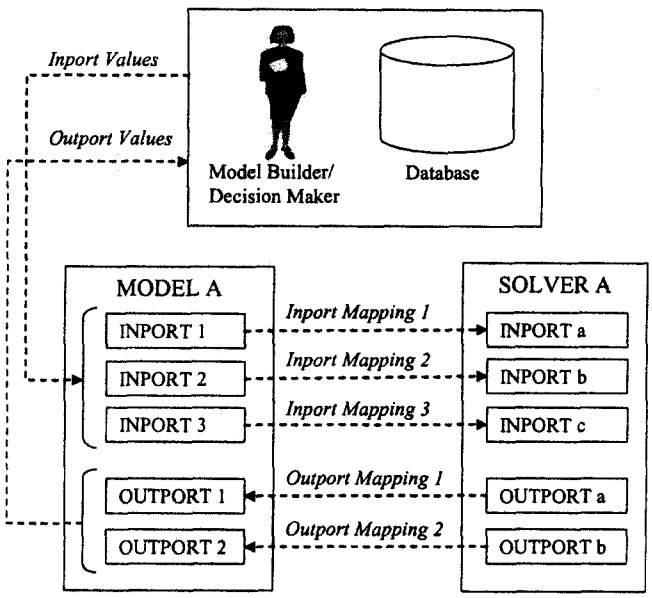


Figure 1 – Model, Solver, and their Interaction.

Meanwhile, a *solver*, characterized by a solver type (SolverType), focuses on the actual problem solving computation, and is responsible for generating the computation results of the modeled decision problem. The solver type has a name, a set of ports for interfacing with model ports, primitive manipulating operations, and a calculation operation implementing the solving algorithm. The ports of the solver type are also classified into inports and outports; the inports accept input parameters of the calculation operation while the outports hold the output parameters after the calculation operation is executed. In the presence of a model, the inports of the solver correspond to the inports of the model and the outports of the solver correspond to the outports of the model. That is, after a compatible solver is selected for model solution, the model delegates its outports to the solver by assigning its inport values to the corresponding inports of the solver. The solver then produces the outport values using its calculation operation and returns them back to the outports of the model. As such, the model solution process requires individual ports of the model to correspond to the solver ports in a one-to-one relationship. We call such a relationship a **port mapping** (Figure 1).

In the following section, we describe an option pricing example of a financial MMS to illustrate these representations for models, solvers, and their interactions in more practical viewpoints. Through the example, the modeling conflict issues, a critical problem for the model-solver integration, are also explained.

An Option Pricing Example

Consider the financial MMS that supports a user's evaluation of financial products by calculating their various numerical analysis factors, such as the net present value (NPV) [15]. In the financial MMS, for user-friendly and intuitive calculations, each financial product is formulated as a model composed of a set of declarative modeling language statements capturing the properties of the product and the analysis factors to be calculated. Additionally, a set of pricing algorithms are also provided as solvers for these financial product models. To exemplify such product models in the financial MMS, we consider two kinds of products: a stock option and a currency option [15]. Figure 2 shows an example model for the stock option using the structured modeling language (SML) [11].

On the other hand, as a solver for the two models, a well-known pricing algorithm called Black-Scholes method [15] is provided, and it requires the following three inports to calculate the NPV: an exercise price, a maturity date, and a dividend rate. First, the *exercise price* is the prearranged price at which the holder of an option can buy or sell the underlying product. The exercise price is also called a *strike price*. Second, the *maturity date* is the date at or by which the underlying product can be bought or sold. Usually, it is represented as a specific date or as a time interval between the present date and the maturity date. The Black-Scholes method solver uses the latter representation. Third, the *dividend rate* is the rate per annum at which a continuous dividend is paid by the underlying stock of a stock option. However, in a real situation, since the dividend of a stock is paid discretely and irregularly, the Black-Scholes method solver needs proper conversion from the discretely paid dividend amounts into the continuously paying rate. In case of a currency option, the dividend rate is analogous to the interest rate of the underlying currency because the currency pays interests as a stock pays dividends. Since the interest rate itself is a continuously paying rate, such a conversion for the stock option's dividends is not required for the currency option's interest rate.

According to the model and solver representation scheme shown in Figure 1, this option pricing example can be illustrated as Figure 3. The financial MMS maintains compatibility between the two models (STOCK_OPTION and CURRENCY_OPTION) and the solver BLACK_SCHOLES by managing their port mappings shown in Figure 3, and thus it can facilitate suggestion of the solver for the two models and solution of them by matching their ports with the corresponding ones of their compatible solver, BLACK_SCHOLES. In this capacity, however, from the solver's perspective, the models have semantic and/or syntactic conflicts with one another in their ports [8,18], and such conflicts require the port mappings to be customized for each model. Typically, the following three types of modeling conflicts are observed among the models:

- The *semantic conflict* implies that depending on the problem semantics of the individual models, the port mappings vary and thus a solver needs to look up different ports in each model to get its inport values. In Figure 3, the ports of STOCK_OPTION and CURRENCY_OPTION that correspond to the port DIVIDEND_RATE of BLACK_SCHOLES are different from each other. In case of STOCK_OPTION, the port DIVIDEND should have a mapping with DIVIDEND_RATE of BLACK_SCHOLES while in case of CURRENCY_OPTION, the port INTEREST_RATE should.
- The *naming conflict* arises when models use synonyms to describe the same ports, or homonyms for different ones. The port EXERCISE_PRICE of STOCK_OPTION and the port STRIKE_PRICE of CURRENCY_OPTION are synonyms. Both ports need to have a mapping with the port EXERCISE_PRICE of BLACK_SCHOLES.
- The *structural conflict* arises when the same ports in individual models are represented in different data formats. The port DIVIDEND of STOCK_OPTION

```
&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.
```

Figure 2 – An Example Model for STOCK_OPTION (Structured Modeling Language).

represented as a matrix (i.e., a list of the two values: dividend paying dates and dividend amounts) should be converted to a single number following the data format of the port DIVIDEND_RATE of BLACK_SCHOLES (i.e., a number as a rate). However, the port INTEREST_RATE of CURRENCY_OPTION represented as a number can be delivered to DIVIDEND_RATE of BLACK_SCHOLES directly.

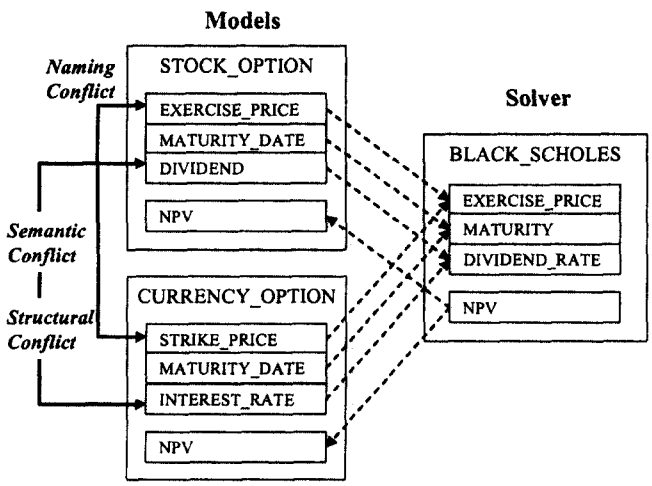


Figure 3 – An Option Pricing Example.

Regarding these three modeling conflicts, the MMS should specify not only the adequate port mappings between individual models and their compatible solvers (for resolving the semantic and naming conflicts) but also suitable data format conversion methods in each port mapping (for resolving the structural conflict). In this paper, such port mappings and data format conversion methods are called **interfacing rules**.

In the following sections, we explain how the model-solver integration framework proposed in this paper resolves these three modeling conflicts by managing the interfacing rules for port-mappings and data format conversion between the two models (STOCK_OPTION and CURRENCY_OPTION) and the solver BLACK_SCHOLES. Based on the interfacing rules, we also present how the framework facilitates the autonomous solver suggestion and intelligent model solution capabilities.

Model-Solver Integration Framework

The model-solver integration framework has two primary tasks: management of the interfacing rules and execution of the solver suggestion and model solution. To support these two tasks, the port mapping dictionary exists as an information registry to manage the interfacing rules for the customized port mappings and data format conversion methods between models and solvers. Referring to the port mapping dictionary, the model and solver agents are placed to perform the autonomous solver suggestion and intelligent model solution, and resolve the semantic, naming, and structural conflicts.

Model Class

A model class is an abstract data structure realizing the model type. Since the model type has a set of ports as its external interface for interacting with its outside environment (i.e., human modelers, solvers and databases), the model class aggregates the port class, implementing the port type. Figure 4(a) shows an object-oriented schema for the model class represented in Unified Modeling Language (UML) [4].

In Figure 4(a), the model class and the port class are named `Model` and `Port`. To handle a wide range of data formats effectively, `Port` can transform into several sub-classes including `NumberPort`, `StringPort`, `DatePort`, `ListPort`, and `MatrixPort`. In determining the data format of a port, two factors are specifically considered: the data domain (e.g., number, string, and date) and the data cardinality (e.g., scalar value, list, and matrix). For the data domain, `StringPort`, `NumberPort`, and `DatePort` are to represent a string, a number, and a date value in the attribute `Value`, respectively. For the data cardinality, `ListPort` and `MatrixPort` are to capture a list or a matrix of values by containing multiple `Port`. Based on the class diagram, Figure 4(b) presents an object data example for the model STOCK_OPTION of Figure 3 using the object diagram of UML.

Solver Class

The solver class (`Solver`) realizing the solver type has a set of ports for interfacing with the model ports. To facilitate solver capability, `Solver` provides three operations, `SetPort()`, `Calculate()`, and `GetPort()`. By calling these operations, the MMS can communicate with a `Solver` object to set inport values, execute the model solving calculation, and get an outport value, respectively.

The `Solver` class is further specialized into sub-classes to support the solver-specific algorithms of individual solvers. Each sub-class representing a particular solver inherits all the attributes and operations from `Solver`, and customizes the operation `Calculate()` by implementing its solver-specific algorithm. Such class inheritance from `Solver` to the sub-classes leads to effective division of tasks and responsibilities between the two. `Solver` defines the general external structure dedicated to the model interfaces (i.e., the aggregated `Port` class) and manipulation operations for MMS communication; the sub-classes focus on the solver-specific algorithm implementations needed for the individual solvers.

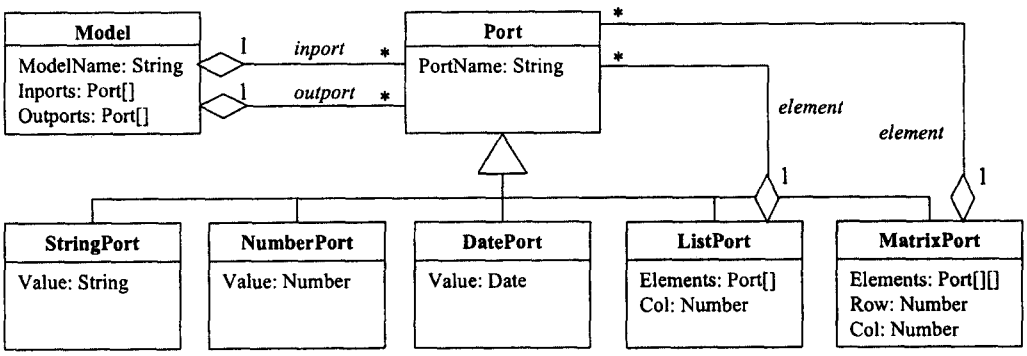
These inheritance mechanisms between `Solver` and its sub-classes are specifically useful when a variety of solvers exist in the solver library since they facilitate the MMS to manage those solver objects in a unified way using the three seemingly generic commands (`SetPort()`, `Calculate()`, and `GetPort()`) provided by `Solver`, regardless of their individual algorithm implementations.

Port Mapping Dictionary

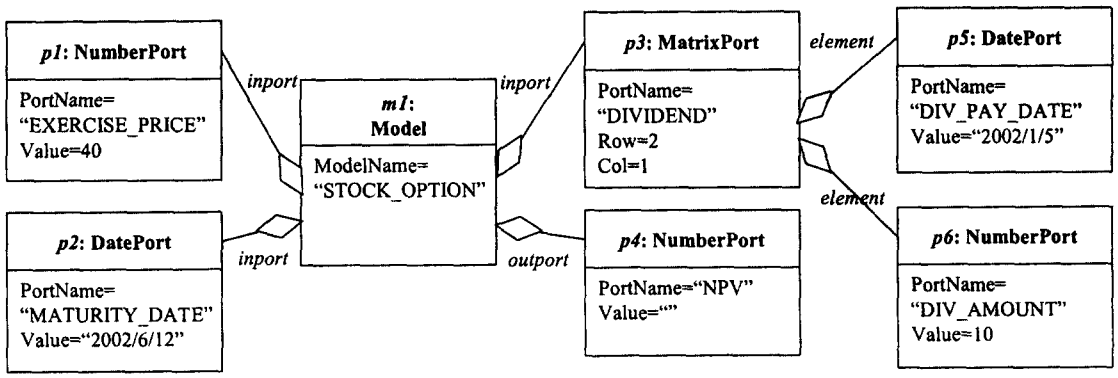
As mentioned earlier, the MMS should manage the interfacing rules for the adequate port mappings and the data format conversion methods between individual models and their compatible solvers. The **port mapping dictionary**, as a dedicated tool for the interfacing rule management, maintains them in a tabular form where each row represents an individual interfacing rule that specifies the port pairs to be mapped and their data format conversion method. Typically, every row of the dictionary has the six attributes (i.e., columns): *model*, *solver*, *model port*, *solver port*, *port type*, and *conversion script*. The attributes *model* and *solver* indicate the model and solver names that the interfacing rule is applied to; the attributes *model port* and *solver port* indicate the corresponding port names of the model and solver to be linked; the attribute *port type* determines whether it is for an inport or an outport; finally, the attribute *conversion script* describes the detailed instructions of the data format conversion.

Specifically, *conversion script* is expressed as an algebraic formula built on two kinds of operands, port references and environmental variables. A port reference means a port value and is denoted by a port name enclosed by ampersands (&) at both ends (e.g., &EXERCISE_PRICE&); an environmental variable refers to a pre-defined system variable and is denoted by a capital string (e.g., TODAY for the current date). These operands can be recursively composed by a set of operators. The operators include domain casting operators for converting the data domain of a port (e.g., TO_NUM for conversion to integer and TO_STR for conversion to string), aggregation operators for converting the data cardinality of a port (e.g., SUM for summation of a list and AVG for average of a list), and other mathematical operators for various numerical calculations (e.g., LOG for logarithm and SQRT for square root). Figure 5(a) shows a conceptual structure of the port mapping dictionary with the option pricing example of Figure 3 where two models (STOCK_OPTION and CURRENCY_OPTION) and one solver BLACK_SCHOLES are under consideration.

Figure 5(b) presents example commands showing how the MMS can use the port mapping dictionary, named `PortMappingDictionary`, in conducting the autonomous solver suggestion and intelligent model solution. The first command shows how to retrieve the compatible solver names of the model STOCK_OPTION from the port mapping dictionary. Since the compatible solvers must have interfacing rules with the model in the dictionary, this command finds the solvers by extracting the attribute *solver* from the interfacing rules that have the model name "STOCK_OPTION" in the attribute *model*. Thus, using this command, the MMS can identify and suggest the compatible solvers autonomously by referring to the port mapping dictionary without any direct user intervention. The second command retrieves the interfacing rules for the inports between the STOCK_OPTION model and the BLACK_SCHOLES solver from the port mapping dictionary. Using the interfacing rules obtained by this command, the MMS can match the model ports with the solver ports



(a) A Schema for the Model Class (Class Diagram of UML)



(b) An Object Data Example for STOCK_OPTION (Object Diagram of UML)

Figure 4 – Model Class Definition.

adequately and convert the data formats of model ports if they are different from those of the corresponding solver ports. The third command converts the data format of the port DIVIDEND according to the conversion script `script1` obtained from the port mapping dictionary. The converted port (`sp`) returned by this command, will be further linked to the corresponding port of the solver (i.e., DIVIDEND_RATE of BLACK_SCHOLES).

Model and Solver Agents

On the basis of the classes `Model`, `Solver`, and `PortMappingDictionary`, two software agents, model agent and solver agent, are defined at a higher level to perform the autonomous solver suggestion and intelligent model solution. Referring to the port mapping dictionary, these two agents cooperate with each other for exchanging port values between models and solvers while resolving the three modeling conflicts.

The **model agent**, which interacts with the user views directly, assists a user's model-solving activities by providing two types of information to the user view: compatible solver names of the model to be solved and the results of the model solution. First, to provide the compatible solver names, the model agent looks them up in the port mapping dictionary using such commands as shown in Figure 5(b). These solver names are then displayed in the user view, and the user selects an appropriate one among the solver names depending on the user's problem-solving purposes. Second, once the user selects a solver and makes a request for the model solution, the model agent sends a model-solving request message to the solver agent. This message consists of the solver name to be used and the inport values required for the solver to execute its calculation operation. In creating the message, the model agent consults the port mapping dictionary to understand how to generate the inport values of the solver by referring to the interfacing rules between the model and the solver. Afterwards, when the solver agent returns the results of its model solution, the model agent provides them to the user through the user view.

In this context, the **solver agent** performs the model solution and returns its results to the model agent in response to the model-solving request message from the model agent. When the solver agent receives the model-solving request message, it retrieves the solver specified in the message from the solver library, fills the inports of the solver with the inport values contained in the message, and executes the calculation operation of the solver. This solver execution process can be done by such commands as shown in Figure 5(b). Once the calculation of the solver is finished, the solver agent sends a model-solving result message to the model agent. In sending the message, as the model agent does to create the model-solving request message, the solver agent also consults the port mapping dictionary to understand the interfacing rules for the outport mappings between the model and the solver.

model	solver	model port	solver port	Port type	conversion script
STOCK_OPTION	BLACK_SCHOLES	EXERCISE_PRICE	EXERCISE_PRICE	Inport	
STOCK_OPTION	BLACK_SCHOLES	MATURITY_DATE	MATURITY	Inport	&MATURITY& = (&MATURITY_DATE& - TODAY) / 365
STOCK_OPTION	BLACK_SCHOLES	DIVIDEND	DIVIDEND_RATE	Inport	&DIVIDEND_RATE&=LOG(1+SUM(&DIVIDEND&[*][2]) / &STOCK_PRICE&)
STOCK_OPTION	BLACK_SCHOLES	NPV	NPV	Outport	
CURRENCY_OPTION	BLACK_SCHOLES	STRIKE_PRICE	EXERCISE_PRICE	Inport	
CURRENCY_OPTION	BLACK_SCHOLES	MATURITY_DATE	MATURITY	Inport	&MATURITY& = (&MATURITY_DATE& - TODAY) / 365
CURRENCY_OPTION	BLACK_SCHOLES	INTEREST_RATE	DIVIDEND_RATE	Inport	
CURRENCY_OPTION	BLACK_SCHOLES	NPV	NPV	Outport	

(a) A Conceptual Structure of the Port Mapping Dictionary for Option Pricing Example

```
/* Querying compatible solvers with a model */
select distinct r.Solver
from r in PortMappingDictionary.rules
where r.Model = "STOCK_OPTION";

/* Querying interfacing rules */
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"
```

```
/* Converting data format of a port */
// Suppose that script1 is a conversion script
// for converting data format of the port
// DIVIDEND of STOCK_OPTION
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) Example Commands for Using the Port Mapping Dictionary

Figure 5 – Port Mapping Dictionary Definition.

Figure 6 shows detailed steps of the solver suggestion and model solution processes performed by the model and solver agents using the sequence diagram of UML. The model and solver agents perform the solver suggestion and model solution autonomously by communicating with each other and referring to the interfacing rules maintained in the port mapping dictionary. In addition, since the interfacing rules provide the port mappings and data format conversion methods between the individual models and their compatible solvers, the agents can intelligently exchange port values between the two while resolving the three modeling conflicts. By virtue of this autonomous and intelligent characteristic of the two agents, what the user should do to solve a model is only select an appropriate solver among the suggested ones on the user view. This simplicity and user-friendliness of the model-solving activities improves productivity and usefulness of the MMS by allowing even non-professional ordinary users to easily use the diverse sets of models and solvers provided by the MMS.

Conclusions

The provision of an intuitive and user-friendly model-solution process is an important functional requirement for the MMS since the model and solver semantics are usually too complicated for most ordinary users to identify the compatibilities between the models and solvers or to understand their parameter matching patterns. Recognizing this requirement of the MMS, this paper proposes a model-solver integration framework that enables the MMS 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 modeling conflicts (i.e., the semantic, naming, and structural conflicts). Among the advantages attained in the model-solver integration framework, the following three are worthy of attention.

First, by defining the constructs including model, solver, port mapping dictionary, model agent, and solver agent without any restrictions on the modeling paradigms, the framework can serve as a building foundation for MMSs in a wide variety of problem domains with multiple modeling paradigms. In the framework, every model statement or solver parameter is uniformly captured as a port regardless of its data format, and thus the model-solver interactions are considered as data exchanges between the model and solver ports. Based on this standardized model-solver interfacing scheme, the port mapping dictionary, model agent, and solver agent consolidate all the primitive solver suggestion and model solution mechanisms generically applicable to diverse sets of models and solvers.

Second, since the interfacing rules between models and solvers are not hard-coded into the MMS and are managed in the port mapping dictionary, the framework can easily

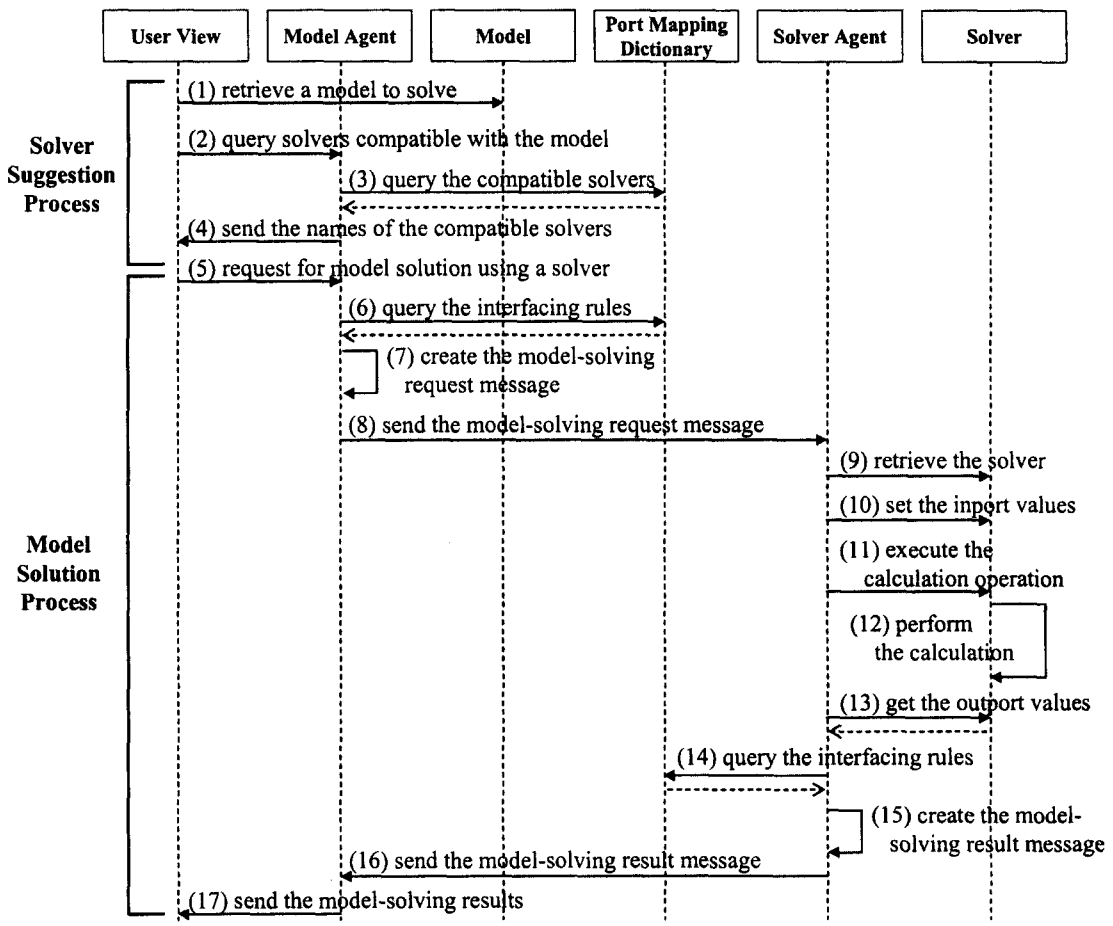


Figure 6 – Solver Suggestion and Model Solution Processes (Sequence Diagram of UML).

adapt to the changes in the models and solvers. When a new model or solver is added to the MMS or an existing one is modified, the framework can additively accommodate and make it functional with other existing ones only by updating the related interfacing rules in the port mapping dictionary without any re-implementation or re-compilation of the entire system.

Third, the autonomous and intelligent characteristics of the model and solver agents make a user's model-solving activities very streamlined and user-friendly. The model and solver agents perform the autonomous solver suggestion and intelligent model solution while hiding the complicated interactions between the models and solvers from the user. Thus, without having precise knowledge of model-solving procedures, users can experiment and solve a model with multiple problem solving purposes using diverse sets of solvers.

A prototype system for the model-solver integration framework has been developed with JAVA programming language with the sponsorship from the Institute of Information Technology Assessment in Korea. Specifically the current research is extended into the distributed computing environment focusing on the Web Services based systems integration platform [5].

Acknowledgements

This research was supported by University IT Research Center Project.

References

[1] Banerjee, S. and Basu, A. (1993). "Model Type Selection in an Integrated DSS Environment," *Decision Support Systems*, Vol. 9, pp. 75-89.

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

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

[4] Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Indianapolis, MA: Addison-Wesley.

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

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

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

[8] Eck, R.D., Philippakis, A., and Ramirez, R. (1990). "Solver Representation for Model Management Systems," *Proceedings of the 23rd Hawaii International Conference on Systems Sciences*, pp. 474-483.

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

[10] Geoffrion, A.M. (1989). "The Formal Aspects of Structured Modeling," *Operations Research*, Vol. 37 pp. 30-51.

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

[12] Goul, M., Philippakis, A., Kiang, M.Y., Fernandes, D., and Otondo, R. (1997). "Requirements for the Design of a Protocol Suite to Automate DSS Deployment on the World Wide Web: A Client/Server Approach," *Decision Support Systems*, Vol. 19, pp. 151-170.

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

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

[15] Hull, J.C. (1997). *Options, Futures, and Other Derivatives*. Upper Saddle River, NJ: Prentice-Hall.

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

[17] Muhanna, W.A. (1994). "SYMMS: A Model Management System That Supports Model Reuse, Sharing, and Integration," *European Journal of Operational Research*, Vol. 72, pp. 214-243.

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

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

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

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

[22] Zeleznikow, J. and Nolan, J.R. (2001). "Using Soft Computing to Build Real World Intelligent Decision Support Systems in Uncertain Domains," *Decision Support Systems*, Vol. 31, pp. 263-285.