Model and Solver Integration For Interoperability Between Options and Their Evaluation Algorithms In Financial Decision Support Systems

Keun-Woo Lee and Soon-Young Huh
Graduate School of Management
Korea Advanced Institute of Science and Technology

Abstract
As the financial option markets grow, financial decision support systems need an efficient framework that enables us to mix and match options and their various evaluation algorithms according to diverse pricing and evaluation purposes. Under such many-to-many relationships between options and algorithms, the systems often encounter inefficiency problems caused by duplicated developments of their interfaces when adding or modifying the options or the algorithms. To resolve such problems, this paper proposes a system framework for dynamic integration of options and algorithms based on the model management system perspective. Specifically, the detailed mapping relationships between the parameters of options and those of algorithms are managed with a separate mapping table, so the addition or the modification is easily incorporated only with altering the table without affecting the integration processes.

Keywords: Model Management, Model and Solver Integration, Financial Decision Support Systems, and Financial Options

1. Introduction

As financial markets grow, financial options [6, 17] have become increasingly important with the contingent properties suitable to meet the market needs being diverse and advanced. Since huge volumes of options are traded outside exchanges [17], financial institutions that are the participants in trading the options can make the options more complicated or create new ones if need be. On the other hand, financial researchers are introducing various evaluation algorithms to calculate theoretical prices and predict future-time market values of the options [17]. Thus, a variety of options and algorithms exist in the markets and they are also continuously evolved. In order to manage such properties of the options and the algorithms, financial decision support systems (DSS) accommodating the two and providing accuracy, timeliness, and easiness in the computation have been widely adopted. In terms of model management systems (MMS) of DSSs, a real option can be conceptualized as a model that specifies the properties and behaviors of the option [13] and an evaluation algorithm as a solver that is invoked and analyzes options when the options need to be analyzed [7,13]. In this capacity, we define options and evaluation algorithms as *option models* and *option solvers*, respectively.

Proliferation and evolution of option models and solvers pose a new challenge to management of them in financial DSSs. This is because it is often the case that a single option solver (e.g., Black-Scholes method) is used in multiple option models and, conversely a single option model (e.g., stock option) also needs multiple option solvers (e.g., Black-Scholes method, Binomial method, and Finite Difference method) due to diverse pricing and evaluation purposes [17]. Moreover, as a new option solver is created, it has to be adopted incrementally without requesting the redevelopment or recompilation of the whole financial DSS. By the same token, when a new option model is introduced, it has to be immediately supported by reusing existing option solvers.

In traditional MMSs, an option model should provide its information as pre-defined parameter values (i.e., value order and data type) to an associated option solver and then the option solver returns the result as a pre-defined parameter value. For example, to compute the theoretical price of a stock option (option model) using the Black-Scholes method (option solver), four parameters of the stock option including exercise price, market price of the underlying asset, volatility of the market price, and maturity date should be passed to the Black-Scholes method. Then, these parameters should be ordered and have data types according to the pre-defined parameter passing rules from the stock option to the Black-Sholes method. However, when a new option solver is created in the MMS and it requires more information from related option models than other existing option solvers, then the option models should be modified accordingly in order to support the new option solver. The more frequently new option solvers are created, the more seriously the maintenance cost should be considered.

To overcome such difficulties, in this paper, we propose a model and solver integration framework and facilitate interoperability between option models and option solvers. Specifically, we address the following three issues, which also constitute the definition of the interoperability: (1) How can the financial DSS make it possible to dynamically mix and match option models with option solvers for satisfying various needs of individual end users? (2) How can an option model or an option solver be newly added or modified into the financial DSS without requiring any modification of the financial DSS? (3) When a new option model (option solver) is added into the financial DSS, how can it be integrated with existing option solvers (option models) without any modification of the existing ones? In addressing these questions, we first define core constructs constituting the integration framework that can be applied to all kinds of option models and option solvers. With these generally defined constructs, we can implement a financial DSS that can manage continuously created or modified option models and option solvers. In presenting system procedures of the architecture, we especially offer a dynamic parameter transformation method to gear a financial DSS towards the interoperability between option models and option solvers.

The paper is organized as follows. In section 2, we briefly review the literatures on integration of models and solvers. In section 3, we define the core constructs for option models and option solvers, and address the integration scheme based on the constructs. In section 4, we provide a few system procedures needed to manage and integrate the option models and solvers with the defined constructs. Finally, in section 5, we summarize our research contributions and propose future research plans.

2. Literature Review

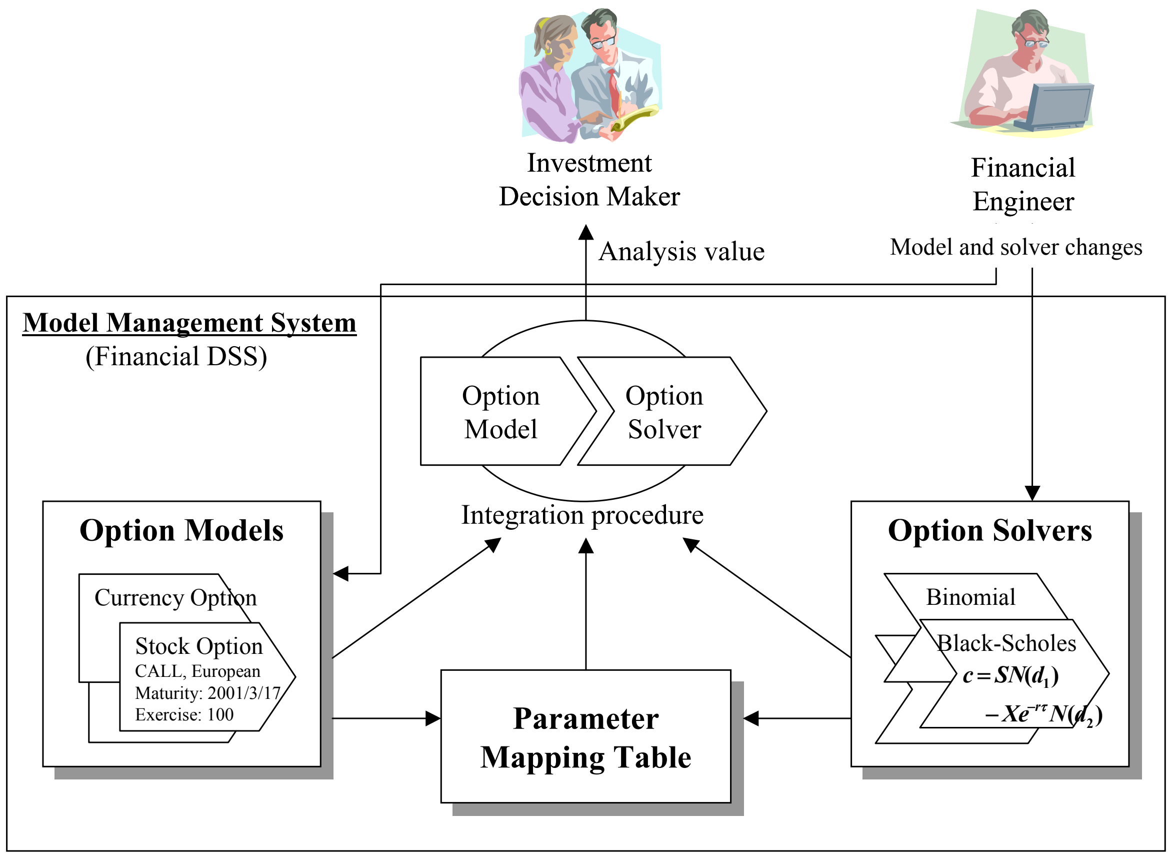
Early researches for the model management system (MMS) focused on the model representation [1,3,10,13] and the independence of data and model [13,21]. They aimed for effective modeling of problems, model reusing, and model sharing. As the research goes on, solving procedures for models are separated from the models to solvers, and implies that a user can solve a single model with multiple problem-solving purposes and apply different models to the same solver [12]. To properly achieve benefits from the model-solver independence, appropriate integration procedure of them is required. Three approaches can be referenced for the integration of models and solvers.

The first approach encodes solvers into a software program at a development time to provide a problem-solving software package in MS/OR domains. Then, an end user imports a model into the program, selects an appropriate solver among available ones of the program, and gets a result by executing the solver [10,13]. Though this approach supports the integration of models and solvers, it still has a serious maintenance problem since the entire software program should be modified and recompiled to add new solvers. Thus, when the program is widely dispersed, it is quite difficult, sometimes impossible, for end users to add solvers containing their proprietary evaluation algorithms for their specific needs. Moreover, this approach lacks applicability for complex problems such as option evaluation algorithms since it deals with only well-defined problems having fixed structures such as a linear programming.

The second approach adopts object-oriented methodologies to achieve flexible management of models and solvers [15,21,22,23]. In this approach, the existing models and solvers can be reused or extended easily due to the polymorphism and inheritance properties of the object-oriented methodologies. Also, the encapsulation property provides an inherent integration method of models and solvers since an object representing a model contains solvers corresponding to the model as its member operations. However, this approach assumes that each solver corresponds to only one model but this assumption is not true as stated in the previous section. Thus, logically same solvers may exist in multiple objects and such redundancy brings about a maintenance problem. That is, when a new model is added, solvers that can be applied to the model should be duplicated in the model object. And a new solver is also duplicated for all applicable models.

The third approach also uses object-oriented methodologies and attempts to facilitate dynamic integration of models and solvers at a run time by separating models and solvers as different objects [8,29]. Specifically, Zhang and Sternbach [29] provided detailed object-oriented designs based on the design pattern [11] to show the applicability of this approach in financial domains. However, in this approach, since the rules for parameter passing that are used to integrate models and solvers are not flexible, a newly created model or solver should obey these pre-defined rules to ensure the interoperability with existing solvers or models. Thus, despite the support of the dynamic integration at a run time, this approach still has limitations. In this sense, the three approaches are only partially successful in supporting the integration of models and solvers, but are still unsatisfactory in the interoperability between them.

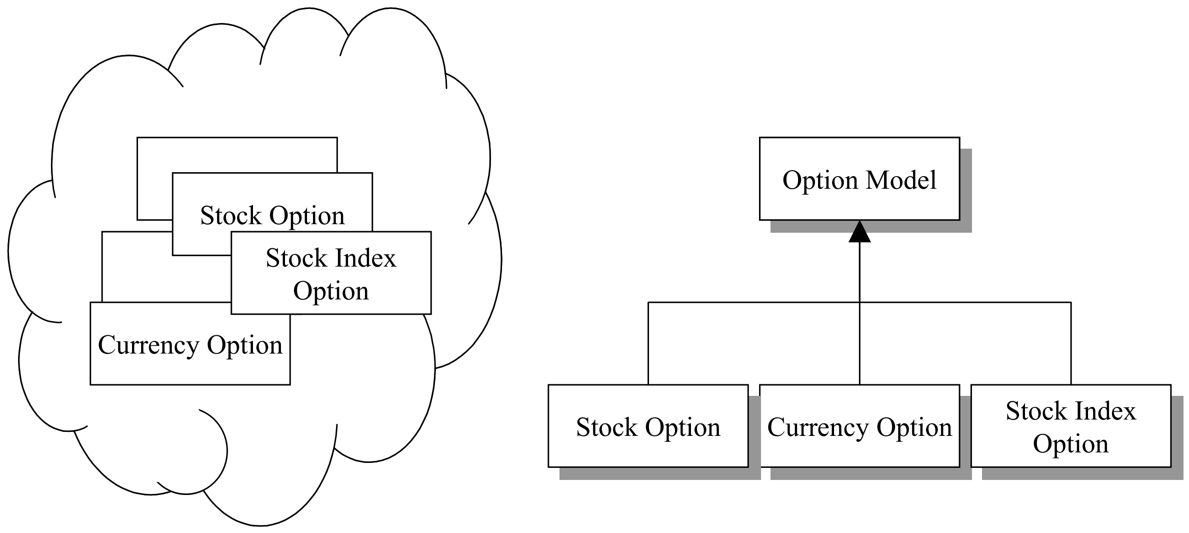
[Figure 1] Conceptual Framework for the Integration of Models and Solvers



3. Core Constructs For The Integration Framework

In this section, we investigate the structures of core constructs constituting the proposed integration framework. [Figure 1] shows the conceptual framework with the constructs. As the base components, the option models and the option solvers are defined, and the financial engineer manages them. For supporting the investment decision maker, the models and the solvers are integrated dynamically on the problem solving requests. In doing this, detailed parameter mapping information for each specific model and solver pair is supplied from the parameter mapping table.

[Figure 2] Inheritance Hierachy of Option Models



(a) A Variety of Financial Products (b) Hierarchy of Option Models

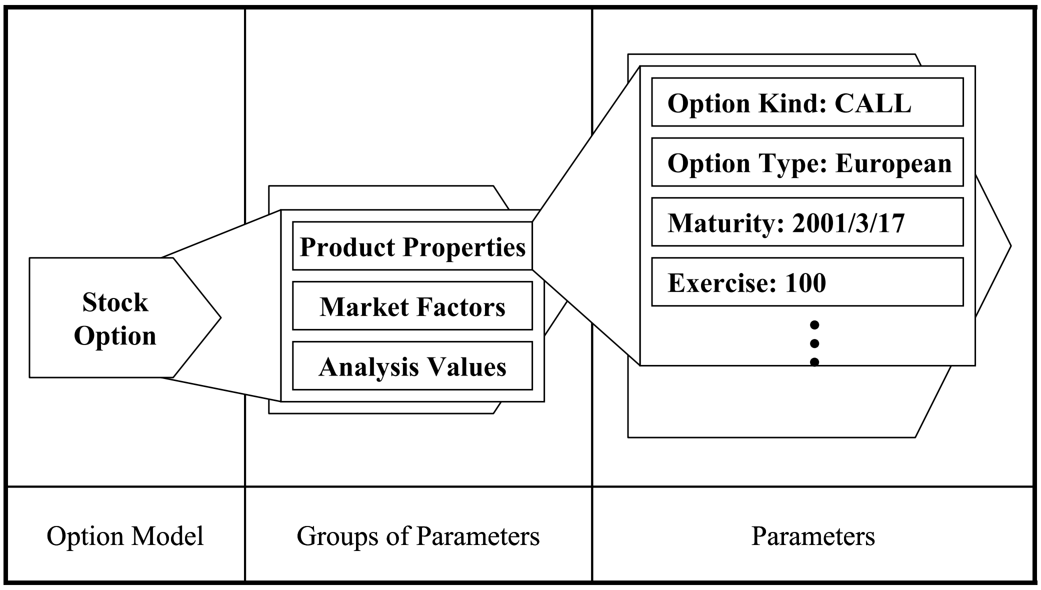
3.1 Option Models and Option Solvers

The first step to achieve the interoperability between models and solvers is representing a variety of option models and option solvers in a uniform way. We can get this uniformity using the inheritance mechanism in object-oriented programming. [Figure 2] shows an example of inheritance hierarchy of a few option products. In the top of the hierarchy, an option model is defined as an abstract model and identifies a general structure that can be uniformly applied to all the option products. It has all the structural cornerstones to preserve various data for all option products, and a set of operations such as management of financial market conventions, manipulation of business days, and so on. Option models representing specific option products and actually used by the system (e.g., Stock Option, Currency Option, and Stock Index Option in [Figure 2]) are inherited from the abstract option model.

Thus, the abstract option model becomes the uniform representation of all option products.

In order to define the generic structure of the abstract option model and build the inheritance hierarchy, we identify all option models as sets of parameters. [Figure 3] shows this structure with a stock option example. The objective of option models in the financial DSS is calculating various evaluation values that are needed for the investment decision makers. Such evaluation values include net present values (NPV) and a few sensitivity indicators [17], and they are calculated from the properties of the option and general market factors such as interest rates and exchanges rates. For example, suppose the financial DSS calculates NPV of a stock option that is a kind of option product upon a stock as its underlying asset, the NPV is calculated from the properties of the option including option kind (put or call), option type (European or American), exercise price, and maturity date, and also from the market interest rates that are general market factors. Thus, we classify the attributes of an option model into three categories: properties of the corresponding product, general market factors, and evaluation values of the product. Based on this identification, the abstract option model provides a generic structure for these parameter group and elemental parameters.

[Figure 3] Structure of Option Models

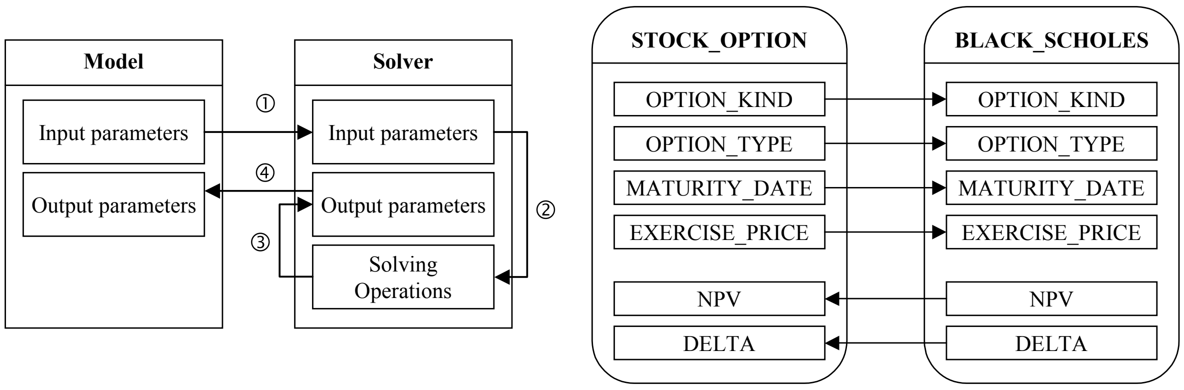


On the other hand, option solvers take the input values of option models and calculate the analysis values that are needed by the clients. Since the input values and analysis values are parameters of the solvers, we can also consider option solvers as sets of parameters like the models. Thus, an inheritance hierarchy of option solvers similar to that of models is defined. In the top of the solver hierarchy, the abstract option solver provides all concrete solvers (e.g., Black-Scholes method, Binomial method, etc.) with the general structures for the parameters. Thus, for the solvers, the uniform representation is the abstract option solver.

3.2 Model-Solver Integration Scheme and Parameter Mapping Table

Based on the structures of option models and solvers, we investigate the model and solver integration procedures in detail. The integration is divided into four steps: transfer of input parameters to the solver, execution of solving operations in the solver, obtainment of solving results in the solver, and transfer of output parameters to the model. The classification of input and output parameter is based on whether a parameter is an input value or output value of the solving operations. In order to clarify the parameter relationships between models and solvers, we apply the classification to not only parameters of solvers but also those of models. [Figure 4(a)] shows the integration procedure. When the system solves a model, the data values in input parameters of the model are transferred to the input parameters of a solver that is assigned to the model (①). Then the solver calculates its output values using solving operations taking its input parameter values as the inputs for the operations (②), and its output parameters hold the result values of the operations (③). And finally, the output parameter values of the solver are returned to the output parameters of the model (④).

[Figure 4] Model and Solver Integration Scheme

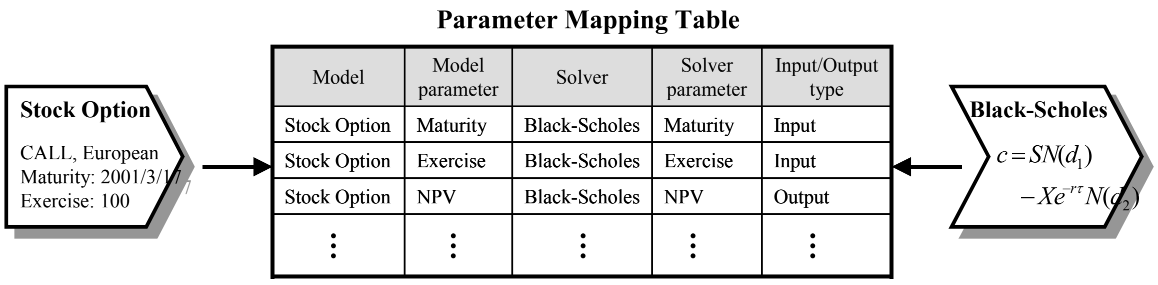


(a) Model and solver integration

(b) Individual parameter mappings between STOCK_OPTION and BLACK_SCHOLES

The individual parameter relationships in this integration procedure are presented in [Figure 4(b)] with an example model and solver, STOCK_OPTION and BLACK_SCHOLES. The parameter relationships are obvious and straightforward to understand the interaction scheme between models and solvers. However, when we try to incorporate changes that can occur in option models or option solvers, a few issues arise to be answered. For example, if a financial DSS supporting STOCK_OPTION and BLACK_SCHOLES currently and start to support a new option solver, BINOMIAL, that can evaluate the STOCK_OPTION using Binomial method, then the STOCK_OPTION should be changed to be able to interact with the new solver like the way with the BLACK_SCHOLES. Since we cannot confirm that the parameter relationships between the STOCK_OPTION and the BINOMIAL are same as those of the STOCK_OPTION and the BLACK_SCHOLES, we should implement every relationship for each model and solver pair. These duplicated implementation problems also occur when a new option solver is added or modified. Due to these problems, additional construct, *parameter mapping table* is defined. Its role is separating detailed mapping information between model parameters and solver parameters from the integration procedures so that the procedures are independent of the kinds of option models and solvers.

[Figure 5] An Example Context of the Parameter Mapping Table



Model	Model parameter	Solver	Solver parameter	Input/Output type
Stock Option	Maturity	Black-Scholes	Maturity	Input
Stock Option	Exercise	Black-Scholes	Exercise	Input
Stock Option	NPV	Black-Scholes	NPV	Output
⋮	⋮	⋮	⋮	⋮

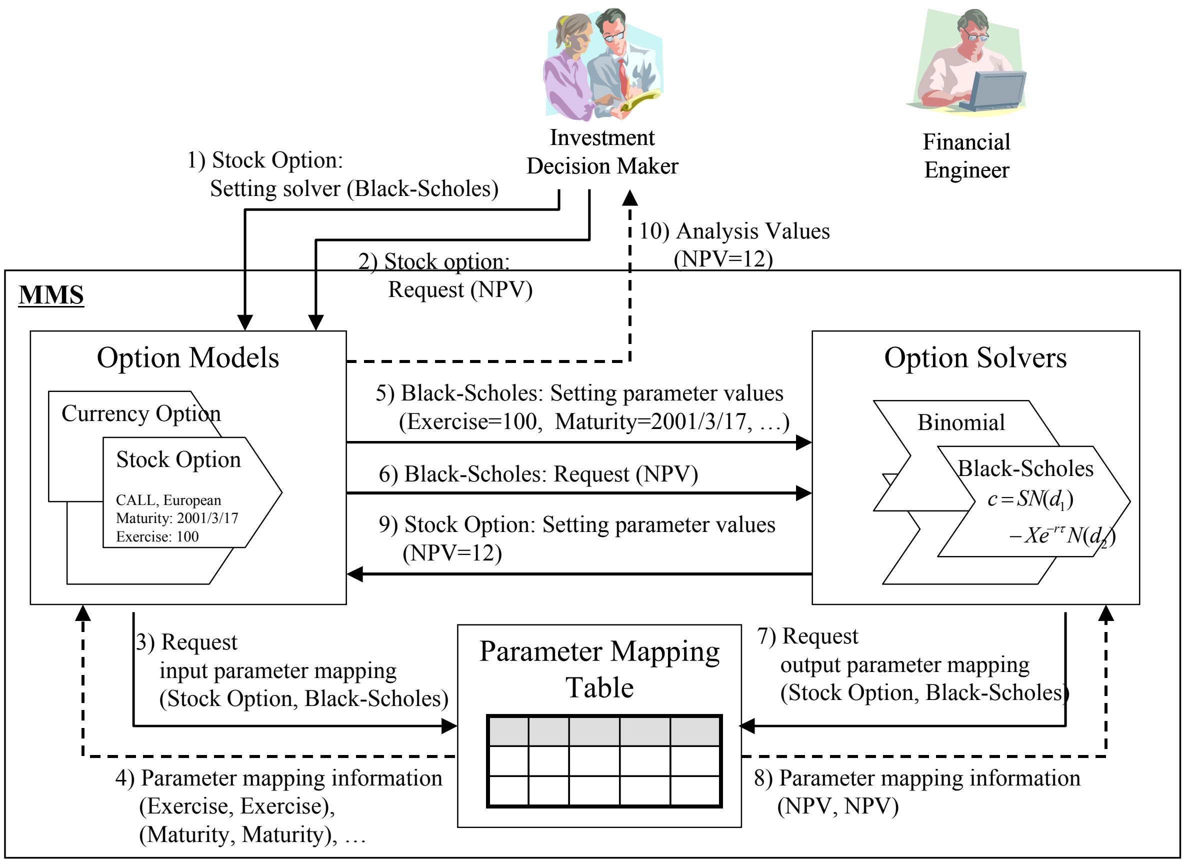
Parameter mapping information is dynamically registered and removed when a new option model or a new option solver is added or an existing one is changed. The parameter mapping table is a dedicated tool set for managing such dynamically changing parameter mapping information in a general and efficient way. [Figure 5] shows the conceptual structure of the parameter mapping table with a stock index option and Black-Scholes example. The parameter mapping table maintains model-solver pair to be matched, parameter pair of the matched model and solver, and the input/output type. With the table, we can get the following three advantages. First, we can make the integration procedure independent of individual parameter mapping information. Second, management of parameter mapping information can be treated like management of data in database system tables, so manipulation of parameter mapping information becomes very easy and familiar for the system users. Last, the existence of the parameter mapping information indicates legitimation of an integration of certain model and solver pair, and identifies interface context of the integration.

Upon the structure of the parameter mapping table, three types of primary manipulation operations are provided: addition, deletion, and retrieval of mapping information for a specific model-solver pair. All transfers of parameter values between option models and option solvers use the retrieval operations to get the mapping information, so the retrieval operations are base operations for the interoperability between models and solvers. Introduction of a new model or a new solver can be incorporated into the integration framework with the addition operations by adding all the necessary parameter mapping information into the parameter mapping table. Modification of the existing parameter mapping information due to changes of models or solvers is also reflected through adding related parameter mapping information or deleting unnecessary mapping information.

4. System Procedures of the Integration Framework

In this section, we supply system procedures operating on the integration framework developed until now. The procedures include integrating models and solvers, adding new models or solvers, and changing solvers.

[Figure 6] System Procedures for Integrating Models and Solvers

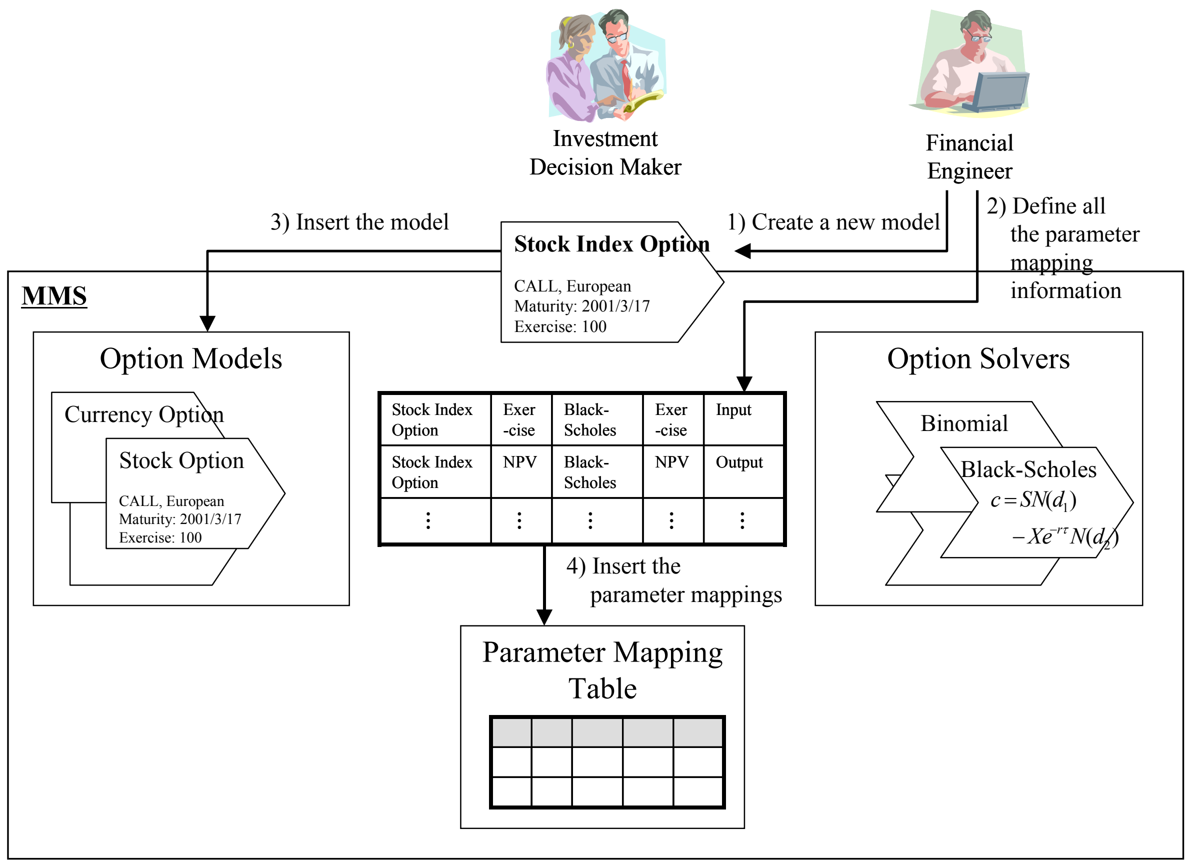

Investment Decision Maker
Financial Engineer
1) Stock Option: Setting solver (Black-Scholes)
2) Stock option: Request (NPV)
10) Analysis Values (NPV=12)
MMS
Option Models
Currency Option
Stock Option
CALL, European
Maturity: 2001/3/17
Exercise: 100
Option Solvers
Binomial
Black-Scholes
$c = SN(d_1) - Xe^{-r\tau}N(d_2)$
5) Black-Scholes: Setting parameter values (Exercise=100, Maturity=2001/3/17, …)
6) Black-Scholes: Request (NPV)
9) Stock Option: Setting parameter values (NPV=12)
3) Request input parameter mapping (Stock Option, Black-Scholes)
Parameter Mapping Table
7) Request output parameter mapping (Stock Option, Black-Scholes)
4) Parameter mapping information (Exercise, Exercise), (Maturity, Maturity), …
8) Parameter mapping information (NPV, NPV)

4.1 Integrating Model and Solver

[Figure 6] shows an example process where the investment decision maker solves the NPV of the Stock Option with the Black-Scholes method. First, 1) he sets the solver Black-Scholes, and 2) requests the NPV output parameter value. Then, 3) the option will get all the parameter mapping information from the parameter mapping table. 4) Upon the received information, only the input parameter values of the model are retrieved and 5) assigned into the corresponding input parameters of the solver. After setting all the input values needed to solve the model, 6) the model requests the NPV on the solver. After solving the NPV of the option, the solver 7) request the parameter mapping information from the parameter mapping table for the output parameters holding the solved results. 8) Receiving the parameter mapping information, 9) the output parameter values of the model are assigned and 10) the NPV is informed to the investment decision maker.

As shown in the example, all the processes working in the integration framework are general and perfectly independent of the specific parameter mapping information. Thus, the processes can be applied to any kinds of option models and option solvers, and adaptable to any changes occurred in the parameter mapping information, namely, the interfaces between the models and the solvers.

[Figure 7] System Procedures for Adding a New Option Model



4.2 *Adding New Model*

[Figure 7] shows an example process where the financial engineer adds a new option model, the Stock Index Option, and updates the mapping information came from the addition. For the new model, all the operations should be done by the financial engineer are only adding it and updating the parameter mapping information properly. 1) He creates the new model and 2) makes appropriate parameter mappings for the models and legitimate solvers. Then, 3) he inserts the new model into the model repository and 4) does the parameter mappings into the parameter mapping table. After that, the newly inserted model is integrated with the solvers in exactly same way with other models as described in [Figure 6].

Similarly with the addition of new model, addition of new solver and modification of existing solver are performed without affecting the integration procedures except the parameter mapping table.

5. Conclusions

In this paper, we propose a system framework for integrating option models and solvers to facilitate the interoperability between them. We define core constructs as base building blocks for the framework generically to embrace various option models and option solvers. In doing this, we consider option models and option solvers as sets of parameters, and integration of them as parameter mapping relationships. With the generalized option models and option solvers, the parameter mapping table managing their specific parameter matching information is also defined for the framework to be independent of the kinds of option models and option solvers. The parameter mapping table takes charge of adding, deleting, and updating the sets of individual parameter mapping information. Based on these three constructs, the integration framework can perform dynamic and type-less integration of option models and option solvers.

In future research, the intent is to focus on two avenues: refinement of the core constructs and resolution of parameter type mismatch. Refinement of the core constructs will include the elaboration of the structures and concrete representation with a formal object-oriented modeling language such as the UML [2]. Resolution of parameter type mismatch is geared to match parameters that have different data types or domains.

REFERENCES

[1] Blanning, R.W. "An entity-relationship approach to model management," *Decision support Systems* (2), 1986, pp. 65-72

[2] Booch, G., Rumbaugh, J., and Jacobson, I. *The unified modeling language user guide*, Addison Wesley Longman, 1999.

[3] Brennan, J.J., and Elam, J.J. "Understanding and validating results in model-based decision support systems," *Decision Support Systems* (2), 1986, pp. 49-54

[4] Chang, A.-M., Holsapple, H.C., and Whinston, A.B. "Model management issues and directions," *Decision Support Systems* (9), 1993, pp. 19-37

[5] Chou, S.T. “Migrating to the web: a web financial information system server,” *Decision Support Systems* (23:1), 1998, pp. 29-40

[6] Copeland, T.E., and Weston, J.F. *Financial theory and corporate policy*, 3rd ed., Addison-Wesley, Reading, MA, 1992.

[7] Eck, R.D., Philippakis, A., and Ramirez, R. “Solver representation for model management systems,” *Proceedings of the 21st Hawaii International Conference on Systems Sciences*, 1990, pp. 474-483

[8] Eggenschwiler, T., and Gamma, E. “ET++SwapsManager: using object technology in the financial engineering domain,” *Proceedings of the conference on object-oriented programming systems, languages, and applications*, Vancouver, B.C. Canada, 18-22 Oct., 1992, pp. 166-177

[9] Fedorowicz, J., and Williams, G.B. “Representing modeling knowledge in an intelligent decision support system,” *Decision Support Systems* (2), 1986, pp. 3-14

[10] Fourer, F., Gay, D.M., and Kernighan, B.W. “A modeling language for mathematical programming,” Management Science (36:5), 1990, pp. 519-554

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design patterns: elements of reusable object-oriented software*, Addison-Wesley, Reading, MA, 1995.

[12] Geoffrion, A.M. “An introduction to structured modeling,” *Management Science* (33:5), 1987, pp. 547-588

[13] Geoffrion, A.M. “The formal aspects of structured modeling,” *Operations Research* (37:1), 1989, pp. 30-51

[14] Hiller, F.S., and Gerald, J.L. *Introduction to operations research*, 5th ed., McGraw-Hill, 1990.

[15] Huh, S.-Y. “Modelbase construction with object-oriented constructs,” *Decision Science* (24:2), 1993, pp. 409-434

[16] Huh, S.-Y., and Chung, Q.B. “A model management framework for heterogeneous algebraic models: object-oriented database management systems approach,” *Omega, Int. J. Mgmt Sci.* (23:3), 1995, pp. 235-256

[17] Hull, J.C. *Options, futures, and other derivatives*, 3rd ed., Prentice Hall, NJ, 1997.

[18] Kruglinski, D.J. *Inside visual C++*, 3rd ed., version 4.0, Microsoft Press, 1995.

[19] Lawrence, M., and Sim, W. “Prototyping a financial DSS,” *Omega, Int. J. Mgmt Sci.* (27:4), 1999, pp. 445-450

[20] Lenard, M.L. “An object-oriented approach to model management,” *Decision Support Systems* (9), 1993, pp. 67-73

[21] Liang, T.-P. “Integrating model management with data management in decision support systems,” *Decision Support Systems* (1), 1985, pp. 221-232

[22] Ma, J. “An object-oriented framework for model management,” *Decision Support Systems* (13:2), 1995, pp. 133-139

[23] Ma, J. “Type and inheritance theory for model management,” *Decision Support Systems* (19:1), 1997, pp. 53-60

[24] Pindyck, R.S., and Rubinfeld, D.L. *Econometric models and Economic forecasts*, 3rd ed., McGraw-Hill, 1991.

[25] Pritchard, J. *COM and CORBA® side by side: architectures, strategies, and implementations*, Addison-Wesley, Reading, MA, 1999.

[26] Raghu, R., and Gehrke, J. *Database Management Systems*, 2nd ed., McGraw-Hill, 2000.

[27] Rizzoli, A.E., Davis, J.R., and Abel, D.J. “Model and data integration and re-use in environmental decision support systems,” *Decision Support Systems* (24:2), 1998, pp. 127-144

[28] Turban, E. *Decision Support and Expert Systems*, Prentice Hall, 1995.

[29] Zhang, J.Q., and Sternbach, E.J. “Financial software design patterns,” *Journal of Object-Oriented Programming* (8:9), 1996, pp. 6-12