A Valuation Technology for Product Development Options Using an Executable Meta-modeling Language

Benjamin H. Y. Koo*,[1], Willard L. Simmons[2], and Edward F. Crawley[2]

[1] Tsinghua University, Beijing, P. R. China
[2] Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

Abstract. Mistakes or foresight in the earlier phases of product development tend to be amplified over the course of a project. Therefore, having a rigorous approach and supporting tools to identify and filter a development portfolio at the early stages can be highly rewarding. This paper presents an executable specification language, Object-Process Network (OPN), that can be used by system designers to formally represent the development option space, and automate certain model refinement activities at earlier phases of product development. Specifically, an OPN specification model can automatically enumerate a set of alternative development portfolios. OPN also provides an algebraic mechanism to handle the knowledge incompleteness problems at varying phases of planning, so that uncertain properties of different portfolios can be represented and analyzed under algebraic principles. In addition, it has a recursively defined model transformation operator that can iteratively refine the specification models to simplify or enhance the details of the machine-generated alternatives. A list of successful application cases is presented.

Keywords. OPN, meta-language, Real Options, Algebra of Systems, Model-Driven Software Development

1 Introduction and Motivation

During the earlier stages of product development, limited engineering resources and knowledge incompleteness inevitably introduce a high degree of uncertainty. However, inflexible design decisions made in the earlier phases tend to seal off the future opportunities of the product development project. Therefore, it is beneficial to employ a design analysis method that can include not one, but a set of possible product development options [1, 4, 5]. Real options is often employed to analyze a set of alternative developmental options

* Associate Professor, Department of Industrial Engineering, Tsinghua University, Beijing, China 100084 Tel: +86-10-62792539 ; Email: benkoo@tsinghua.edu.cn

Our paper describes a model-driven analysis method and a supporting tool that extends existing real option analysis methods.

Real options analysis is related, but different from, financial option analysis. When a financial option is purchased, certain rights in the future are contractually protected [7]. Conversely, product development options in the real world usually provide little if any guarantee. For example, the investment in certain technologies may or may not create additional opportunities in the future. Therefore, when modeling real options, the modeling method must deal with this additional level of uncertainty. Furthermore, when comparing between product development alternatives, it is often necessary to preserve the structural and behavioral compositions of the alternative scenarios. Many quantitative option analysis methods assume that the possible behavioral and structural evolutions of the option portfolios of interest can be abstracted into a few statistical measures. To preserve and analyze the structural and behavioral information content in product development options, we utilize modeling principles inspired by Hoare and Cousot [2, 6] to develop a model-driven method for product development option analysis which can preserve the quantitative, quantitative, and fuzzy aspects of "real" options.

2 A Model-Driven Analysis Method

Model-driven methods are software development techniques that compose the resulting model of a software product recursively using other software models as building blocks. It often employs a special-purpose model manipulation language [3] that manipulates and generates different versions of models.

We adopt a model-driven method to analyze engineering product development options because model-driven methods are effective techniques for combination and comparisons of different options. Competing project plans can also introduce uncertainty because they require comparative analysis before a decision can be made. Therefore, when comparing or composing two or more development options, analysts must first encode options and the respective compositional structures as standard *model data types* in a modeling manipulation language. Then, analysts may apply model manipulation operations to analyzed the represented options, such as equality or substitution. Using an executable model-driven language, these analytical operations can be automated to mechanically reason about the logical consistency or other quantitative and qualitative (defined over an un-ordered domain) properties of different options.

2.1 The Modeling Vocabulary

We hereby define three model data types and one operator for the model-driven method. These data types are: $\mathcal{V}_{opt}$, $\mathcal{F}$, and $\mathcal{S}$. They stand for product

development option, payoff function, and product development portfolio respectively. The operator is called $\mathcal{D}$ for product development decision. They are defined as follows:

Definition 1 (Product Development Option: $\mathcal{V}_{opt}$). *$\mathcal{V}_{opt}$ is a data type, in which $v \in \mathcal{V}_{opt}$ is a product development option. An option v represents a set of defined possibilities, whose actual values are to be determined under certain or uncertain real world constraints.*

When v defines a finite number of possibilities [7], it can be treated as a discrete variable. When v contains an infinite number of possible values, the option can be modeled as a symbolic variable, whose actual value can be assigned at the point of observation. For example, an aircraft frame material option can be chosen from a finite set, such as $\{aluminum, wood, steel\}$. The interest rate of a certain loan can be represented as a continuous variable, symbolized by a variable name v_r, where $v_r > 0$.

Definition 2 (Payoff Function : $\mathcal{F}$). *$\mathcal{F}$ is a data type, in which $f \in \mathcal{F}$ is an instance of a payoff function. It relates two or more development options. Each f determines the payoff values or constrains the allowable value combinations of directly related options, v's.*

When all related development options are statically related, the payoff function f can be modeled as a conditional probability function, which represents all possible value combinations and associated distribution of these options. Once any one of these options' values are determined, the payoff function can be used to compute the value distributions for other related options. When two or more options are related temporally, a payoff function can be constructed to take the value of a temporally-causal option and compute the value(s) of the temporally-dependent option(s). For example, $v_{output} = f(v_{input1}, v_{input2}, ...)$. Payoff functions can also be analyzed using algebraic rules to substitute, simplify, or compose into different functions or values.

Definition 3 (Product Development Portfolio : $\mathcal{S}$). *$\mathcal{S}$ is a data type, in which $s \in \mathcal{S}$ is a portfolio. A portfolio is a collection of v's and f's. It may also be written as a tuple, $s = \langle \{v_1, ..., v_j\}, \{f_1, ..., f_k\} \rangle$, where j and k are the number of options and payoff functions respectively.*

A portfolio s is a composition of many product development options. The structure of a portfolio is captured via an associated set of payoff functions that relate these options. Under this definition, a portfolio s can be modeled as a graph containing two domains of elements. One set of elements represent options and the others represent payoff functions. The structures (relationships) between the options and payoff functions can be shown as a bi-partite graph. A bi-partite graph is a highly expressive formalism. It can be used as the basic syntax for many kinds of computationally complete modeling languages [8].

Based on the computational properties of different s's, we may treat them executable specifications that capture the behavioral and structural properties of different compositions of product development options.

One must note that our set-based definition of development option allows the possibility of *recursion*. For example, when there exists multiple competing product development portfolios, these portfolios make up a set that represents an option instance.

Definition 4 (Product Development Decision : $\mathcal{D}$). *A product development decision $\mathcal{D}$ is an operator on the $\mathcal{S}$ domain, or: $\mathcal{D}(s) \in \mathcal{S}$.*

The operator $\mathcal{D}$ is a *model refinement function* over $\mathcal{S}$. It takes one or more instances of s as inputs, and produces one or more instances of s by assigning option values or specializing the payoff functions in the output s. For example, by choosing to build an aircraft frame using aluminum material, the decision operator $\mathcal{D}$ may replace the material option with three choices ($\{aluminum, wood, steel\}$), into an option with only one choice ($\{wood\}$). One should note that $\mathcal{D}$ is a meta-operator, whose effects on the domain $\mathcal{S}$ is driven by the inputs' information content, also specified using elements in $\mathcal{S}$. For example, the *encode*, *enumerate*, and *evaluate* operations on $\mathcal{S}$ (explained further in Section 3.1) can be thought of as specialized versions of $\mathcal{D}$.

2.2 Identify portfolios using fixed-point search

As defined earlier, a design decision $\mathcal{D}$ is closed over the domain of portfolios. It modifies the information content of a portfolio by adding or removing a collection of real options or payoff functions. The design process as a whole can be thought of as an iterative procedure that applies design decisions to successive revisions of an initial portfolio. Therefore, the evolutionary history of a product development portfolio, s, can be formulated as a fixed point formula:

$$s_{n+1} = \mathcal{D}_n(s_n)$$

Where s_0 is an initial portfolio that is made up of many related development options and associated payoff functions. The term n indicates the sequential index number of decisions made to change the portfolio. The functional operator $\mathcal{D}_n$ is the nth version of the design decision operator. The design decision operator $\mathcal{D}$ may have many versions because at each point a design decision is made, it is mostly likely to have a different effect on its operand since its behavior is partially defined by s_n. One can view this dynamic definition of $\mathcal{D}$ as a way to implement *dynamic programming*.

This fixed-point formulation of product design process assumes that all design information can be encoded in the domain of portfolios, $\mathcal{S}$. When a fixed-point in the $\mathcal{S}$ domain is found ($s_{n+1} = s_n$), we arrive at a *fixed* or *stabilized* portfolio. This *fixed* portfolio can be considered for implementation in the real-world. This fixed-point formulation of portfolio refinement reflects the dynamic nature of portfolio refinement in practice.

3 An Executable Modeling Tool

The above-mentioned method requires a modeling tool that enables one to encode, enumerate, and evaluate development portfolios. The rationale of implementing the tool is twofold. First, the above method involves certain tedious model manipulation tasks that must be automated. Second, it should serve as an experimental prototype to determine whether this type of automated model construction tool can be useful in real-world engineering projects. This section briefly describes the functional features and high-level software architecture of a tool, Object-Process Network (OPN). OPN can be characterized as an executable meta-language designed for model manipulation tasks [8, 9].

3.1 Functional Areas

There are three main functional areas of the tool: *encode*, *enumerate*, and *evaluate* models that represent option portfolios. They are three specialized versions of the meta-operator $\mathcal{D}$ mentioned earlier.

The first functional area is a model editor that enables users to *encode* a portfolio as a collection of development options and a collection of payoff functions. This model editing feature is implemented as a bi-partite graph editor. It allows users to insert and delete options (represented as boxes) and payoff functions (represented as ellipses). A screenshot of the editor is shown in Figure 1. There are specialized editors for development options and payoff functions that allow users to specify allowable values and function definitions for these design primitives. The editor also enables users to insert and remove edges between the boxes and ellipses. Edges are data structures that carry boolean expressions to determine whether certain value combinations are acceptable. Each bi-partite graph diagram shown in Figure 1 represents a development portfolio.

The second functional area is to *enumerate* all the possible variations of the development portfolio. For statically related options, a portfolio can be treated as a Graphical Game model, which can be approximated solved using Bayesian Belief Network algorithms [8]. The enumeration procedure for temporally dependent options is realized via a directed graph enumeration algorithm illustrated in detail in Reference [8]. These algorithms assess the probability distribution of likely sub-portfolios or generate a collection of portfolio variations.

The third functional area is to *evaluate* the properties by human or by machine. During the enumeration process, the algorithm for enumeration also carries out the calculations of payoff functions, yielding additional information about options, or eliminating options that are not compatible. Since each payoff function may assess more than one property, there are usually multiple named properties associated with each portfolio. These named properties can be used as multi-dimensional metrics to compare different portfolios.

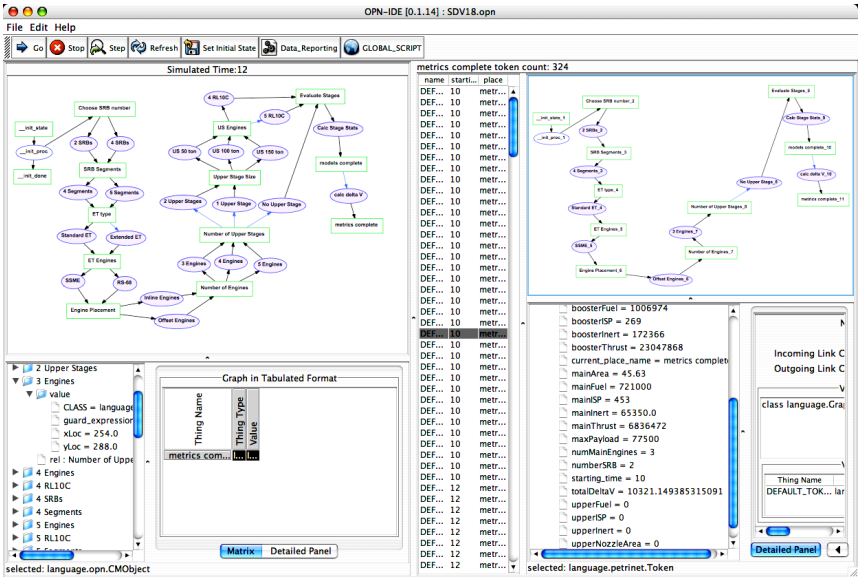

Fig. 1. Screen capture of the OPN tool.

3.2 Software Architecture

To support the three functional areas, the codebase is organized into three software packages. They are `LanguageKernel`, `PersistenceServices`, and `UserInterfaceWidgets`. They are all implemented in Java.

`LanguageKernel` includes four basic data types: Option, Payoff, Relationship, and Portfolio. The data type called *Relationship* implements the constraints specified in the payoff functions ($\mathcal{F}$). A portfolio is implemented as a bi-partite graph. The decision operator, $\mathcal{D}$ is implemented as a Java method that merges or deletes certain parts of the bi-partite graph. The graph editing, enumeration, and function evaluation algorithms are all implemented as specialization of the decision operator. These algorithms always return the results in the portfolio data type. This implementation strategy is intentionally designed to mimic the nature of an algebraic system, so that we may inductively reason about the computational results of this software library.

`UserInterfaceWidgets` is the package that provides the graphical user interface that allows users to visualize and edit portfolios in a number of graphical modes. The user interface of OPN is intentionally designed to allow users visualize that both data content as well as structural properties of development portfolios are being manipulated. Moreover, it shows that portfolio refinement is an iterative procedure that starts from an initial portfolio and unfolds into many generations of alternative portfolios. The generated portfolios are listed in the middle section of the editor and can be individually selected and edited. Each of the portfolios is also associated with a set

of properties listed in the lower right corner of the editor window. Users can *decide* between different *portfolio options* as a part of the human-in-the-loop decision procedure.

`PersistenceServices` is a software package currently implemented to store portfolios as XML files. This package is designed to allow future extensions that utilizes scalable database services when users need to deal with a much larger number of generated portfolios.

4 Applications

This method and its supporting tool, OPN, have been successfully applied to study varying compositional structures of different product development portfolios and assess the interactions between many qualitative and quantitative variables. Due to limitations on article length, the following list briefly summarizes three published applications:

- A study of Moon and Mars exploration architectures for the NASA Vision for Space Exploration [10]. In this study, over a thousand alternative, feasible mission-mode options were generated and compared for human travel to the Moon or Mars.
- A study of developmental options for flight configurations of a particular type of military aircraft [8]. This study demonstrated OPN's ability to reason about the possibility space of physical configurations under incomplete information.
- A study of options for Space Shuttle derived cargo launch vehicles [11]. This study generated and evaluated hundreds of developmental portfolio options for evolving the Space Shuttle's hardware into a new launch vehicle.

OPN helped streamline the exploration of many combinatorial possibilities in different option portfolios. It also supports numeric calculation of payoff values, and the calculations can be postponed or symbolically simplified without sacrificing the integrity of the analysis results.

5 Discussion and Conclusion

This model-driven approach allows us to deal with three critical problems in product development option analysis. The are listed as follows:

1. Enable product development planners to represent the state-space of real options as a network of model-driven data elements. Based on model-driven analysis principles, combinatorially explosive option state-space maybe systematically partitioned into a manageable number of sub-spaces each represented by a sub-network model.

2. Provide an executable language to encode portfolios as executable models, composed of a collection of options and their payoff functions. The models can be executed to enumerate alternative combinatorial scenarios and perform various levels of simulation to assess their strengths and weaknesses.
3. Under *incomplete knowledge*, portfolios maybe algebraically manipulated [9] . This approach may proceed by applying algebraic rules to infer certain logical properties of the portfolios. In contrast, many quantitative analysis methods often require observational, statistical or stochastic simulation results before formal analysis may proceed.

This model-driven method tackles knowledge incompleteness problems in option analysis using an algebraic approach. It also preserves the structural and behavioral properties of different product development portfolios during the analytical process. The list of rather different applications of this method, indicates that this model-driven analysis framework maybe applied to a broad range of decision-making problems.

References

1. C. Y. Baldwin and K. B. Clark. *Design Rules, Vol. 1: The Power of Modularity*. The MIT Press, Mar 2000.
2. P. Cousot and R. Cousot. Compositional and inductive semantic definition in fixpoint, equational, constraint, closure-conditioned, rule-based and game-theoretic form. In P. Wolper, editor, *Computer Aided Verification: 7th International Conference*, LNCS 939, pages 293–308. Springer-Verlag, July 3-5 1995.
3. K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison-Wesley, 2000.
4. W. Griswold, M. Shonle, K. Sullivan, Y. Song, N. Tewari, Y. Cai, and H. Rajan. Modular software design with crosscutting interfaces. *IEEE Software*, 23(1):51–60, 2006.
5. J. Guttag and J. J. Horning. Formal specification as a design tool. In *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 251–261, New York, NY, USA, 1980. ACM Press.
6. C. A. R. Hoare. Process algebra: A unifying approach. In J. W. S. Aku E. Abdallah, Cliff B. Jones, editor, *Communicating Sequential Processes*, July 2005.
7. J. C. Hull. *Options, Futures, and other Derivative Securities*. Prentice Hall, 2nd edition, 1993.
8. B. H. Y. Koo. *A Meta-language for Systems Architecting*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2005.
9. B. H. Y. Koo, W. L. Simmons, and E. F. Crawley. Algebra of systems: an executable framework for model synthesis and evaluation. In *Proceedings of the 2007 International Conference on Systems Engineering and Modeling*, 2007.
10. W. L. Simmons, B. H. Y. Koo, and E. F. Crawley. Architecture generation for Moon-Mars exploration using an executable meta-language. In *Proceedings of AIAA Space 2005, 30 August - 1 September, Long Beach, CA*, 2005.
11. W. L. Simmons, B. H. Y. Koo, and E. F. Crawley. Space systems architecting using meta-languages. In *56th International Astronautical Congress*, 2005.