MathOCL: a domain-specific language for financial modelling

Howard Haughton[1], Sobhan Yassipour Tehrani[2] and Kevin Lano[3]

[1]Holistic Risk Solutions Ltd, Croydon, London, UK
[2]University College London, Gower Street, London, UK
[3]King's College London, Strand, London, UK

Abstract

Financial models are mathematical models representing the concepts underlying financial processes such as valuation and risk estimation. Validation of such models is viewed as an essential activity by banking and other financial regulatory authorities. Regulators require institutions to periodically validate financial models to ensure that they are operating as intended. In our view, concepts from model driven engineering (MDE) could be employed to support the model definition and validation process. In this paper, we describe a domain-specific language (DSL) for expressing financial models, and associated tools for analysing these models and for transitioning to an executable implementation which is correct-by-construction with respect to the models.

Keywords

Financial models, Model validation, MDE

1. Introduction

The finance industry is one of the most significant UK industries in terms of employment and contribution to GNP [13]. It also plays a major role in facilitating UK industrial and business activities through the provision of investment and credit services. Underpinning the activities of finance institutions are mathematical theories and models, such as the Black-Scholes option pricing model [6].

Under international and national regulatory requirements, banks/financial institutions are required to ensure that their internal financial models are fit for purpose. Fit for purpose means that they work as expected based on conditions set by regulators. Since market and/or operational conditions change relatively frequently, the possibility exists that internal models might underestimate the actual amount of risks faced by banks. However, even beyond the use of models for regulatory capital, model validation has utility where such models are used to price transactions, make business decisions or report financial results, for example.

In this paper we define a domain-specific language, MathOCL, to support the specification and validation of financial models. Section 2 describes the concepts of a financial model and of model validation. Section 3 discusses the relevance of MDE for financial modelling, and Section 4 describes the proposed MathOCL language, using extracts from a realistic validation case to illustrate the specification and analysis techniques supported by MathOCL and its tools. Section 5 describes related work, and Section 6 gives conclusions.

AMDE 2023: Agile Model-driven Engineering Workshop, Part of the Software Technologies: Applications and Foundations (STAF) federated conferences, Eds. K. Lano, H. Alfraihi, S. Rahimi and J. Troya, 20 July 2023, Leicester, UK.

howard.haughton@gmail.com (H. Haughton); sobhan.tehrani@ucl.ac.uk (S. Y. Tehrani); kevin.lano@kcl.ac.uk (K. Lano)

2. Financial Model Validation Concepts

Although the Basel Committee on Banking Supervision provides general guidance on financial model validation, this is often in relation to specific types of risk such as market [2] or credit risk [3]. Yet there appears to be an absence of a global standard for general financial model validation and little work has been undertaken toward this end. There are, however, various proposed best practices for model validation, see for example [9].

Perhaps the most widely used definition of a (financial) model is credited to the United States Federal Reserve [5]:

> "The term model refers to a quantitative method, system, or approach that applies statistical, economic, financial, or mathematical theories, techniques, and assumptions to process input data into quantitative estimates."

The Federal Reserve definition of model validation is:

> "Model validation is the set of processes and activities intended to verify that models are performing as expected, in line

> with their design objectives and business uses. Effective validation helps to ensure that models are sound, identifying potential limitations and assumptions and assessing their possible impact. All model components—inputs, processing, outputs, and reports—should be subject to validation; this applies equally to models developed in-house and to those purchased from or developed by vendors or consultants."

3. Model-driven Engineering

Model-driven engineering (MDE) can be described as an approach in which high-level abstract software models (e.g., capturing user requirements or designs) are constructed and transformed (using formalised rules) into executable code, see for example [15, 27, 18].

Although MDE has been successful in certain application domains, particularly in aerospace and automotive systems [10, 23], there has been low uptake of MDE in the finance sector, despite the high importance of software correctness and quick time-to-market in this sector. One factor blocking the use of MDE in finance is the distance between the notations used by finance practitioners (the classical mathematics of continuous functions and stochastic processes, together with Excel spreadsheets) and those available in MDE languages such as OCL [24], which focus on discrete mathematics, formal logic and set theory. For individual practitioners, or for businesses, the learning curve and resources needed to utilise MDE languages and tools has usually been too high relative to the expected benefits.

In order to apply MDE to financial model development we have to take into account that model validation involves analysis of the underlying mathematical, statistical, economic and other assumptions. In other words, an MDE approach for financial modelling should support domain-specific constructs.

3.1. Domain-specific languages

A domain specification language (DSL) is a language that has constructs which are capable of representing the idiosyncrasies of a particular problem domain [11, 16]. Just as there is no single model that can be used to represent all problems in finance, there will be no single DSL capable of denoting all financial problems. Consequently, several DSLs are likely to be required to support model specification and validation. As a priority, we focussed upon the following extensions of OCL for financial specification:

- Conventional mathematical notations for integration, $\int_a^b$, differentiation, ∂_x, statistical expectation $E[expr]$, summation (Σ), product (Π) and powers (e.g., e^x).
- The ability to define random variables following the normal distribution and other significant statistical distributions.
- Facilities for algebraic simplification and symbolic evaluation (e.g., of integrals and differentials).
- Facilities to support equation solving and proof documentation.

4. The *MathOCL* DSL for Financial Applications

Following the above principles, a DSL for expressing financial models has been defined based on the OCL specification language [24]. Called *MathOCL*, this provides a **specification** construct to contain financial models, and the constructs of Table 1 to specify financial variables, relations between these, and instructions to solve equations or to simplify formulae. A tool has been defined to support the editing and analysis of MathOCL specifications[1]. Although there are other mathematical DSLs such as [1, 4, 25], MathOCL is novel because it uses classical mathematical notation, instead of a program-like notation. This can provide a more natural means of use for domain experts, at the cost of developing more extensive tool support ([1] and [25] are embedded DSLs, utilising Python and Haskell environments, respectively). In addition, MathOCL is novel in supporting *argumentation*, the presentation of logical steps in the derivation of theorems from a set of model constraints and assumptions.

The instructions $instr$ in the third case in Table 1 can be:

- Expand $expr$ to n terms – for the Maclaurin expansion of $expr$, $n \leq 4$
- Substitute v in $expr$ – substitute the defined value of v for v in $expr$
- Factor $expr$ by v – express $expr$ as $v * fct$ for some expression fct
- Cancel v in $e1/e2$ – simplify $e1/e2$ by cancelling v in numerator and denominator
- Express $expr$ as polynomial in v – express $expr$ as a polynomial in v.

The expression language of MathOCL includes classical calculus and statistical notations. Thus finance models can be expressed in a form which is familiar to analysts.

[1] mathapp.jar at github.com/eclipse/agileuml

Table 1
MathOCL constructs

Construct	*Explanation*
Define v	Introduce variable v
Define $v = expr$	Define v's value as $expr$
Define $v = instr$	Define v's value as the result of $instr$
Define $v \sim D$	Define v as a random variable from distribution D
Simplify $expr$	Simplify an expression
Solve $eqns$ for $vars$	Solve a set of equations: Quadratic, simple differential and multiple linear equations are supported.
Constraint on $v \mid expr$	Constrain v by $expr$
Theorem $expr$ when $expr$	Assume schematic theorem
Prove $expr$ if $assm$	Prove $expr$ from $assm$

4.1. Case study

As an example of financial model validation, we describe some key steps in the validation of a standard approach for pricing share-based options, the Cox, Ross, Rubinstein Binomial model [8], against that of Black-Scholes [6].

The Binomial model assumes that

- The underlying asset does not pay dividends;
- There are no arbitrage opportunities;
- Interest rates are constant over the life of the option and continuously compounded;
- Investors are risk-neutral;
- The market is frictionless in that there are no transaction costs or taxes;
- The underlying stock price is positive and follows a random walk in that it will either move up or down with a certain probability.

Here we consider European options based on shares:

> (European options). A European option is a type of derivative which provides the holder the right to buy (call option) or sell (put option) an underlying asset at a specified time, t, for a fixed price, K. A call option has a payout at time t of $max\,(S_t - K, 0)$ and the put $max\,(K - S_t, 0)$, where S_t is the underlying asset price at time t.

The Binomial model divides the term T of the derivative into a sequence of periods, in which the underlying asset price is modelled as either increasing or decreasing by fixed ratios u, d with probabilities p and $1 - p$, respectively. Assuming the current time is $t = 0$ and the stock price is $S_t = S_0$, the price at $t = 1$ will rise to $S_0 * u$ with probability p or fall to $S_0 * d$ with probability $1 - p$ where $u > 1, 0 < d < 1$. Hence at $t = 1$ we have

$$S_1 = \begin{cases} S_0 * u & with\ probability\ p \\ S_0 * d & with\ probability\ 1 - p \end{cases}$$

Arbitrage is a situation where, with positive probability p, someone could earn riskless profits by taking a long/short position in the stock and also borrowing/lending a corresponding amount of funds in the money markets with a view to replicating an options payoff. To avoid arbitrage, the following condition must be satisfied: $d < e^r < u$ where r is the continuously compounded interest rate, typically assumed to be greater than zero. This guarantees that the probability p is positive and satisfies $0 < p < 1$.

4.1.1. Replicating portfolio

A replicating portolio [26] is a portfolio consisting of holding positions in assets which produce the same set of cash flows as a derivative. In our case, a replicating portfolio for the European option will consist of holdings h_1 units of the stock and h_2 units of cash. Hence at $t = 0$, the value of the portolio is

$$V_0 = h_1 * S_0 + h_2 \tag{1}$$

When $t = 1$, the portfolio value will be

$$V_1 = \begin{cases} h_1 * S_0 * u + h_2 * e^{rt} = O_u & probability\ p \\ h_1 * S_0 * d + h_2 * e^{rt} = O_d & probability\ 1 - p \end{cases} \tag{2}$$

In a multi-period binomial model, the up/down movements would repeat at each period, for example, as depicted in Figure 1.

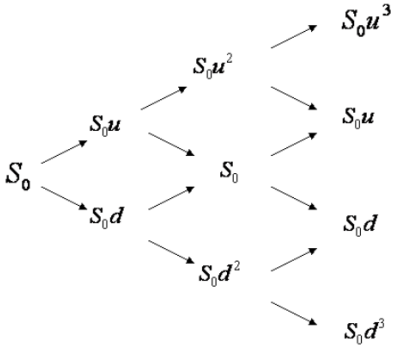

Figure 1: 3-period binomial model

Equation (2) can be expressed in a MathOCL specification of the form shown in the MathOCL toolset in Figure 2.

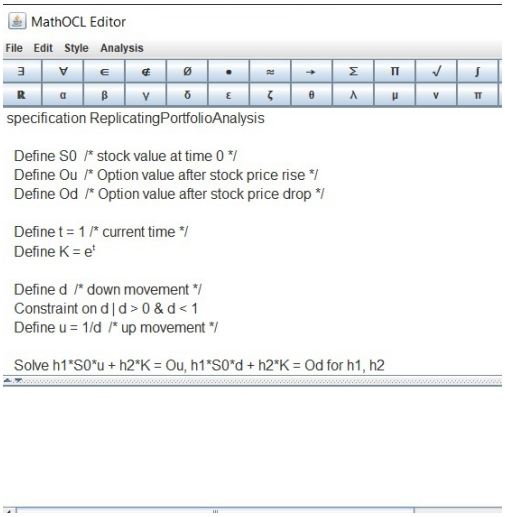

Figure 2: MathOCL model

The equation in this model can be automatically solved (symbolically) using the MathOCL tools, and interactively simplified to produce a more explicit version (Figure 3 shows the computed equation solutions, and the user instructions to apply a cancel operation to simplify the solutions; Figure 4 shows the final simplified solutions). The editor provides interactive support for writing MathOCL specifications, with automated keyword guidance and syntax checking.

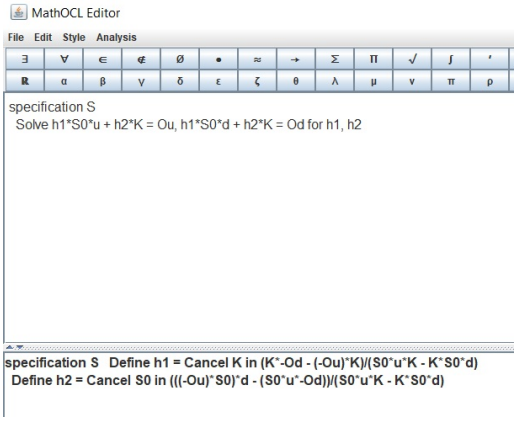

Figure 3: MathOCL model: equation solution, instructions

To define the algebraic simplifier, we used the CSTL/CGTL grammar-based transformation language [21, 22], which operates on the abstract syntax trees of MathOCL expressions. For example, rewrite rules to simplify arithmetic expressions include:

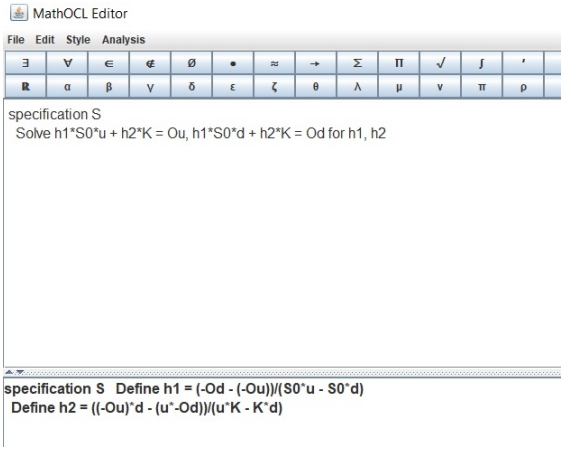

Figure 4: MathOCL model after simplification

```
1 * _1  |-->_1
0 * _1  |-->0
_1 * _1  |-->(_1)^{2}
_1 * _2  |-->_1*_2
```

Conditional rules and auxiliary functions can also be used, for example:

```
_1 = _2 |-->_1‘fArgument = _2‘fArgument<when>
          _1‘isStrictlyIncreasing true,
          _1‘functor _2‘functor
```

This rewrite rule expresses that equations such as $f(e1) = f(e2)$ can be simplified to $e1 = e2$, if f is strictly increasing.

4.2. Theorem-proving and visualisation

The MathOCL tools provide basic proof checking using algebraic simplification facilities, e.g., to deduce $x < y$ from $e^x < e^y$. The tools enable the documentation of manually-constructed proofs and arguments, to facilitate the assessment of these reasoning steps. We intend to extend these facilities to support general proof construction. In the example model validation case, a key argument is that as the number N of periods in the binomial tree for the term T of the option, each period being of length $Dt = T/N$, is increased, then the option price estimate converges to the Black-Scholes price. One stage in this argument is shown in the top panel of Figure 5, which presents individual steps in a chain of reasoning to conclude that $\mu * Dt < \sigma$. The lower panel shows the algebraic simplification of these steps.

4.3. Generation of code

From a simplified and explicit MathOCL specification containing only $Constraint$, $Simplify$ and $Define$

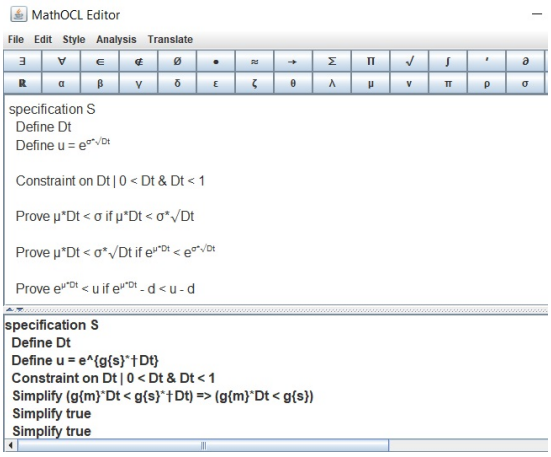

Figure 5: Proof steps in MathOCL

constructs, a computational specification can be automatically generated in standard UML/OCL. The explicit definitions $Define\ v = expr$ are mapped to attribute definitions of v together with an operation computing $expr$. In turn, from this UML/OCL specification, executable code in multiple languages can be generated using the AgileUML toolset[2], including code in Python 3.9, C# and Java (Figure 6 shows the C# code of the replicating portfolio model). Another route to executable code is via the zAppDev tools of CLMS Ltd., and we are defining a bridge translation from MathOCL to the Mamba3 language of zAppDev in order to utilise these tools [14]. In either case, supporting OCL libraries for financial mathematics and Excel functions need to be defined, together with implementations in target programming languages. The specific new libraries are $MathLib$, $FinanceLib$, $OclRandom$ and $Excel$:

- $MathLib$ – routines for matrix operations, numerical root-finding procedures, etc
- $FinanceLib$ – specialised financial functions such as bond pricing, yield estimation and Monte-Carlo simulation
- $OclRandom$ – random number generators for uniform, normal, Poisson and other distributions
- $Excel$ – routines for all Excel Workbook functions, such as the Gamma function, statistical correlation, etc.

The specifications for these can be found at `github.com/eclipse/agileuml/libraries`, together with example implementations in different languages.

We have carried out a number of case studies of financial specification using MathOCL in order to evaluate its

[2] github.com/eclipse/agileuml

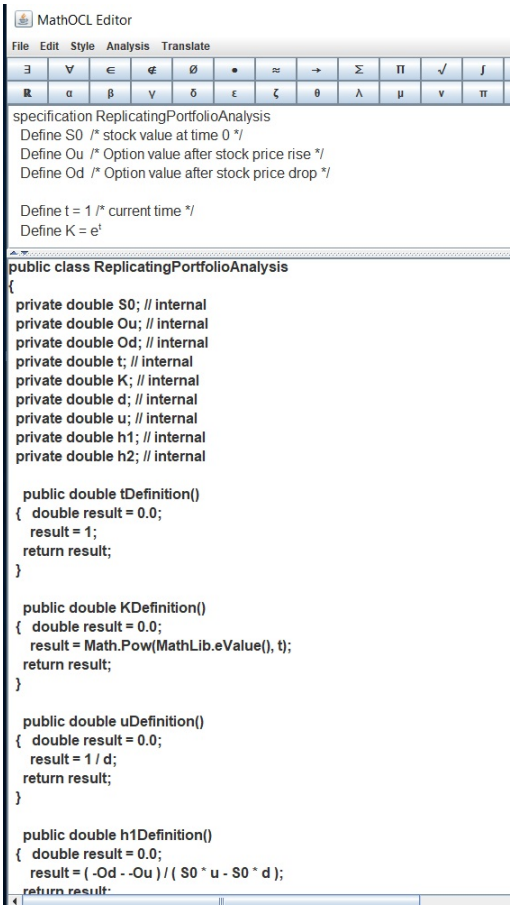

Figure 6: Generation of C# code

effectiveness for financial model definition and analysis. We also compared its capabilities to those of PySym and the Matlab Symbolic Math Toolbox.

5. Related Work

Various approaches for model-based or formally specified financial engineering have been proposed, of which the Kapital system at J. P. Morgan was one of the most successful [7]. Other work has focussed on the definition of languages to declaratively specify financial products ([4, 25]), and not on the modelling of financial *processes*. However process specification is necessary because many financial models concern the design of processes such as Monte-Carlo simulation or other valuation procedures.

Other related work includes languages for real-time specification [12, 17] and for engineering model specification [1]. Standard OCL has also been used for the specification of financial software applications ([19, 20]), but

it does not have the representations for the elements of financial mathematics needed to express financial models, such as integration, differentiation and statistical distributions.

6. Conclusions

In this paper we have considered the application of MDE for financial model specification and validation. We defined the MathOCL DSL to support the definition and analysis of financial models, and showed the application of the DSL on extracts from a realistic model validation case.

Acknowledgments

The MathOCL project is a collaboration between Holistic Risk Solutions Ltd., CLMS UK Ltd, King's College London, Agile MDE Ltd., and University College London. We acknowledge the support of the KCL NMES Enterprise pump-priming fund and UK EPSRC Seedcorn funding from the MDENet network grant. Dr Yannis Zorgios of CLMS UK Ltd and Kaveh Aryanpoo of KCL also contributed to this work.

References

[1] M. Alnaes et al., *Unified form language: a domain-specific language for weak formulations of partial differential equations*, ACM Trans Math Software, 2014.

[2] Basel Committee on Banking Supervision, *Calculation of RWA for market risk*, https://www.bis.org/bcbs/publ/d352.pdf, 2019, (Accessed 1 December 2022).

[3] Basel Committee on Banking Supervision, *Calculation of RWA for credit risk*, https://www.bis.org/bcbs/publ/d457.pdf, 2019, (Accessed 1 December 2022).

[4] S. Branavan, *Technology trends in capital markets*, EXTENT-2016, 2016.

[5] Board of Governors of the Federal Reserve System, Office of the Comptroller of the Currency, *Supervisory Guidance on Model Risk Management*, https://www.federalreserve.gov/supervisionreg/srletters/sr1107a1.pdf, 2011, (Accessed 1 December 2022).

[6] F. Black, M. Scholes, The Pricing of Options and Corporate Liabilities, *Journal of Political Economy*, 8, 637-654.

[7] Cincom, *JP Morgan derives clear benefits from Cincom Smalltalk*, www.cincom.com/pdf/CS040819-1.pdf, 2016.

[8] J. Cox, S. Ross, M. Rubinstein, Option Pricing—A Simplified Approach, *Journal of Financial Economics*, 7, 229-263, 1979.

[9] P. J. de Jongh, et al., A proposed best practice model validation framework for banks. South African *Journal of Economic and Management Sciences*, 20(1), pp. 1–15, 2017.

[10] U. Eliasson, et al., *Agile MDE in mechatronic systems – An industrial case study*, MODELS 2014, LNCS vol. 8767, Springer, 2014.

[11] R. France, B. Rumpe, Model-driven Development of Complex Software: A Research Roadmap, *Future of Software Engineering* (FOSE '07), 2007, pp. 37-54, doi: 10.1109/FOSE.2007.14.

[12] S. Goldsack, K. Lano and A. Sanchez, *Transforming Continuous into Discrete Specifications with VDM^{++}*, IEE C8 Colloquium Digest on Hybrid Control for real-time Systems, 1996.

[13] House of Commons report, https://researchbriefings.files.parliament.uk/documents/SN06193/SN06193.pdf, 2022.

[14] I. Ibrahimi, D. Moudilos, *Model slicing on low-code platforms*, FVPM, 2022.

[15] A. Jilani, M. Iqbal, M. Khan, M. Usman, Advances in Applications of Object Constraint Language for Software Engineering. *Advances in Computers*, Editor(s): Atif M. Memon, Elsevier, Volume 112, Pages 135-184, 2012.

[16] H. Krahn, B. Rumpe, S. Volkel, MontiCore: a framework for compositional development of domain specific languages. *International Journal Software Tools Technolology Transfer*, 12: 353–372, 2010.

[17] K. Lano, S. Goldsack, J. Bicarregui, S. Kent, *Integrating VDM^{++} and Real-time System Design*, Z User Meeting, Reading, UK, Springer-Verlag LNCS vol. 1212, 1997, pp. 188-219.

[18] K. Lano, Agile Model-based Development using UML-RSDS, *CRC Press*, https://doi.org/10.1201/9781315368153, 2017.

[19] K. Lano, H. Haughton et al, *Agile model-driven engineering of financial applications*, FlexMDE, MODELS 2017.

[20] K. Lano, H. Haughton, *Financial Software Engineering*, Springer-Verlag, 2019.

[21] K. Lano, Q. Xue, S. Kolahdouz-Rahimi, *Agile specification of code generators for model-driven engineering*, ICSEA 2020.

[22] K. Lano, Q. Xue, *Lightweight software language processing using Antlr and CGTL*, Modelsward 2023.

[23] S. Mirachi et al., *Applying agile methods to aircraft embedded software*, SPE, vol. 47, 2017, pp. 1465–1484.

[24] OMG, *Object Constraint Language 2.4 Specification*, OMG document formal/2014-02-03, 2014.

[25] S. Peyton Jones, J. M. Eber, J. Seward, *Composing Contracts: an adventure in financial engineering*, ICFP 2000, ACM.

[26] M. Rubinstein, H. Leland, Replicating Options with Positions in Stock and Cash, *Financial Analysts Journal*, Vol. 37, No. 4, pp. 63–72, 1981.

[27] R. Van Der Straeten, T. Mens, S. Van Baelen, Challenges in Model-Driven Software Engineering. M.R.V. Chaudron (Ed.): *MODELS 2008 Workshops*, LNCS 5421, pp. 35-47, 2009.