Model driven engineering in finance: bond valuation, from formulae to code

Asad Rahman[1,*,†], Kevin Lano [2,†] and Howard Haughton[3,†]

[1] *King's College London, 300 Strand, London, United Kingdom*
[2] *King's College London, 300 Strand, London, United Kingdom*
[3] *Holistic Risk Solutions, Croydon, London, United Kingdom*

Abstract
This paper discusses the use of Model Driven Engineering (MDE) in finance for the purposes of defining a mathematical model and then translating it into executable code. The process involved in the formula-to-code translation provides traceable and transparent steps that are useful for educational and audit purposes. This is especially beneficial in the context of regulatory requirements, model transparency, and user understandability. The paper demonstrates the automated process constituents that transform the model firstly into MathOCL, OCL and finally into code. The stages involved in the derivation of the translation process are demonstrated by using a mathematical example of a bond and translating this into Python code..

Keywords
Financial models, Model validation, MDE, DSL, regulatory requirements

1. Introduction

Financial institutions such as banks and insurance companies invest in financial products such as bonds for the purposes of capital growth over a period. This is done to meet future financial obligations and offset current and future financial liabilities for regulatory purposes or simply provide a return on a customer's investment.

Bond investments and returns are modelled mathematically, and the validation of such models is not only essential but is scrutinized by regulators. Model-driven engineering (MDE) can be used to define and validate financial mathematical models by abstracting model complexity into an easily understood context by the end user. Using MDE can result in increased model accuracy and transparency, as well as a quicker time to market for the model. The resulting model can be implemented in a software language of choice such as Python, C# or Java.

[*] Corresponding author.
[†] These authors contributed equally.

asad.rahman@kcl.ac.uk (A. Rahman); kevin.lano@kcl.ac.uk (K. Lano); howard.haughton@kcl.ac.uk (H. Haughton)
0009-0007-2675-1995 (A. Rahman); 0000-0002-9706-1410 (K. Lano); 0000-0002-3159-4329 (H. Haughton)

2. Outline of Objectives

The research seeks to establish a rigorous and systematic process for the development of financial applications from mathematical specifications to executable code. The development process uses the inherent object-oriented modelling of financial products for the purposes of reusability and specialisation. Additionally, the research seeks to create supporting software libraries for a universe of financial instruments. The aim is to provide a platform for the development of applications using the process template going forward.

3. State of The Art

MDE has been used successfully in high-integrity domains such as aerospace and automotive systems (Mohagheghi et al, 2013), however there only limited use cases of MDE in the finance industry (Cincom, 2016), (Nakicenovic, 2012) and (Shirtz et al, 2007). Organisational factors have traditionally limited the uptake of MDE, whereby institutions are opposed to the organisation-wide unification of data and processes (Kumaresan et al., 2018). Limiting technological factors include working practices of using spreadsheets or direct coding approaches, instead of UML and OCL design approaches (Lano et al., 2023), which maybe considered as pre-requisites for using MDE.

The paper (Haughton et al. , 2023) introduces a domain specific language (DSL), namely MathOCL, based on the Object Constraint Language (OCL) (OMG, 2014). MathOCL is intended for the definition of mathematical equations to support financial modelling. OCL on the other hand allows for the definition of financial constraints. Object-oriented characteristics of MathOCL such as inheritance align with financial concepts of the reuse and the specialisation of financial instruments.

4. Methodology

The methodology for exploring the development of financial instruments using MDE was developed in stages, described in the subsequent section below, that resulted in an end-to-end process definition. The process starts from a mathematical model and ends with executable code as shown in the diagram (Figure 1).

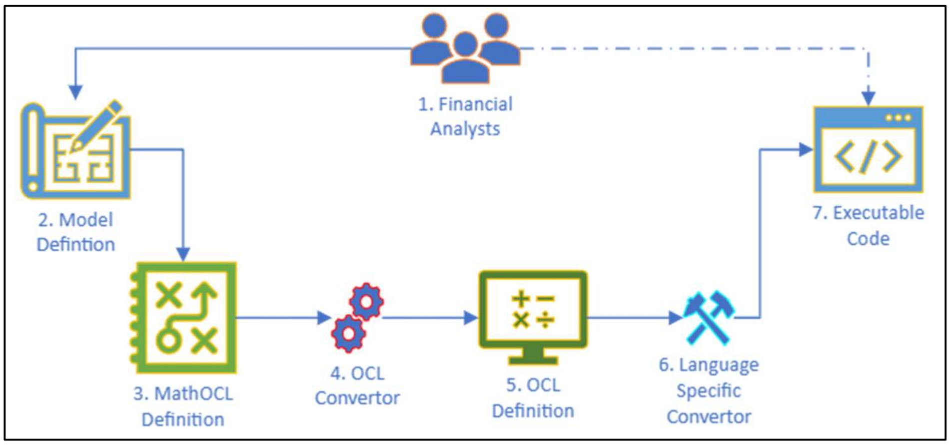

Figure 1: Process Overview. Model Definition to Code.

4.1. Development of Methodology

The research began with a literature review of MDE in finance. This revealed a single example, Kapital system at J.P. Morgan (Cincom, 2016), which was decommissioned due to project complexity. Subsequently, a bond equation was chosen to test its evolution through MathOCL, OCL and finally Python code. Next, the bond price was independently and mathematically verified. Further exploration will follow that will consider other financial instruments such as options amongst others. Finally, model conversion accuracy and efficiency aspects will also be researched further.

5. Bond Implementation and Valuation

Bonds can be represented as a mathematical model to emulate a loan made by an investor to a borrower. A bond type, namely a fixed-rate bond (FRB), provides a return through periodic interest payments with the final return of the principal invested (Murphy, 2023).

5.1. Bond Pricing

The mathematical formula for an FRB is shown below (Kumar, 2018) :

$$Value = \sum_{i=1}^{n} \frac{C}{(1+r)^i} + \frac{F}{(1+r)^n} \tag{1}$$

where ... C = coupon payment, n= number of payments, r = rate, F = face value

5.2. MathOCL

MathOCL as shown by (Haughton et al. , 2023) is a DSL that is based on OCL and is intended for the definition of mathematical equations. The MathOCL representation of the above bond definition is given below:

$$\text{Value} = \Sigma_{i=1}{}^{n}\ C/(1 + r)^{i} + F/(1 + r)^{n}$$

Figure 2: Bond Formula in MathOCL representation

5.3. Object Constraint Language (OCL)

The MathOCL definition is converted into OCL syntax and constraints as mentioned by (Sunitha & Samuel, 2018) are applied to ensure that the supplied parameters are relevant to the calculation being performed. A sample is shown in the diagram (Figure 3) below.

```
context Bond
-- Coupon rate range 0% and 100%
inv CouponIsValid: self.coupon >= 0.0 and self.coupon <= 100.0

-- Yield to maturity; non-negative
inv YieldToMaturityIsValid: self.ytm >= 0.0

-- Coupon frequency; non-negative
inv FrequencyIsValid: self.freq > 0
```

Figure 2: Bond constraints in MathOCL representation

5.4. Derived Code

OCL is translatable into a number of different languages by providing mapping definitions for the syntax of the target language. Constraints are also mapped onto the target programming language equivalent to produce executable code (Hamie, 2004). The diagram below (Figure 4) shows a snippet of Python code.

```
def is_coupon_valid(self):
    return 0.0 <= self.coup <= 100.0

def is_yield_to_maturity_valid(self):
    return self.ytm >= 0.0

def is_frequency_valid(self):
    return self.freq > 0
```

Figure 3: Python code translated from OCL

5.5. Code Validation

The bond calculation was validated using a publicly available open source Python library (bond-pricing (pypi.org, 2022)) in order to confirm that the resulting price is as expected.

6. Conclusion

The paper forms part of an ongoing research project that has so far demonstrated a process that can be used to develop software starting from a mathematical specification of a financial instrument. MDE can thus be used to demonstrate the validation of financial models by demonstrating the accuracy and validity of the underlying mathematical models where the user focuses on what a task should do rather than how it is implemented in code (Fowler, 2010). The universe of financial instruments used in the industry can be covered by further developing the libraries in MathOCL. Similarly, the code can be generated in any programming language of choice by expanding the convertor libraries that generate target code. Performance metrics will further help to evaluate the efficiency and accuracy of the transformation process. This case study can be validated and replicated using the tools and framework surrounding MathOCL and OCL.

References

[1] P. Mohagheghi, Where does model-driven engineering help? Experiences from three industrial cases, 2013.

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

[3] D. Shirtz, M. Kazakov, Y. Shaham-Gafni, Adopting Model Driven Development in a Large Financial Organization, 2007.

[4] A. Kumaresan, D. Liberona, A case study on challenges and obstacles in transforming to a data-driven business model in a financial organisation, pp. 263--276, 2018.

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

[6] M. B. Nakicenovic, An Agile Driven Architecture Modernization to a Model-Driven Development Solution, vol 5, nos. 3, 4, 2012, pp. 308–322.

[7] J. Fernando. Bond: Financial Meaning with Examples and How They Are Priced, 2023. URL: https://www.investopedia.com/terms/b/bond.asp

[8] M. Fowler, Domain specific languages. AddisonWesley, London 2010.

[9] A. Hamie, Translating the Object Constraint Language into the Java, 2004. URL: https://www.researchgate.net/publication/221000520_Translating_the_Object_Constraint_Language_into_the_Java_Modelling_Language

[10] H. Haughton, S. Tehrani, K. Lano, MathOCL: a domain-specific language for financial applications, 2023.

[11] A. Kumar, Bond Pricing Formula, 2023. URL: https://www.wallstreetmojo.com/bond-pricing-formula/#h-example-1

[12] C. Murphy, Fixed-Income Security Definition, Types, and Examples, 2023. URL: https://www.investopedia.com/terms/f/fixed-incomesecurity.asp

[13] OMG, Object Constraint Language 2014, OMG document formal.

[14] pypi.org, bond-pricing, 2023, URL: https://pypi.org/project/bond-pricing/

[15] E. Sunitha, P. Samuel, Object constraint language for code generation from activity models,2023.URL:https://www.sciencedirect.com/science/article/abs/pii/S0950584916304190