Towards Incremental Migration of Industrial Mathematical Models

Marcus Gerhold[1], Martijn Riemeijer[2], Ömer Sayilir[1] and Vadim Zaytsev[1]

[1]Formal Methods & Tools, University of Twente, Enschede, The Netherlands
[2]Thermo Fisher Scientific, Eindhoven, The Netherlands

Abstract

Tools for building mathematical models vary widely across fields and industries, creating communication challenges even between domain experts who rely on different tools within the same company or department. In this paper we showcase a prototype for a multidirectional transformation among mathematical models built in (subsets of) Python, R, and Excel. We introduce a canonical intermediate representation called MIMIR, intended as the single source of truth regarding the details of a mathematical model instance. Changes made in one model type can be synchronised with this intermediate representation, which in turn are propagated to the others.

This prototype is a first step in an ongoing collaboration with Thermo Fisher Scientific to migrate existing executable behaviour models towards a domain-specific language co-designed with domain experts. We illustrate transformations and successive edits across three supported formats, and discuss the restrictions imposed by the prototype's input contracts. These observations inform the ongoing language design and the intended transition from existing modelling artefacts to a shared representation.

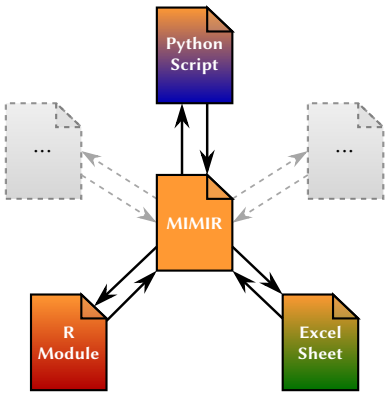
Keywords

Bidirectional Transformations, Mathematical Models, Multidirectional Transformations, Software Migration

1. Introduction

Mathematical models are used in fields such as physics [1], chemistry [2], economics [3], environmental sciences [4] and engineering [5], to model and gain insight into certain behaviour of systems under specific scenarios. There are many tools to build these mathematical models, such as Microsoft Excel [6], Python [7], R [8], MATLAB [9], or Julia [10]. In the industrial context, according to our observations, the diversity of tools used by different people even when working on the same product or project, while allowing employees the appreciated freedom, eventually leads to communication challenges. In some companies, like our industrial partner Thermo Fisher Scientific, operating in the field of cyber-physical systems (CPS), such observations lead to attempts to migrate towards a narrower set of technologies.

In this paper we focus on *behaviour models* which are executable mathematical models of different real processes (physical, chemical, optical, electrical, etc) built as a part of CPS design or maintenance. Our ultimate goal is to migrate existing behaviour models within Thermo Fisher Scientific to a new domain-specific language (DSL) which we are currently co-designing with the domain experts. As a stepping stone towards that goal, we have built an intermediate representation (IR) which allows us to see these behaviour models in three model formats: Excel, Python, and R; plus a set of bidirectional transformations (BX) between this IR and the actually executable artefacts. It is easy to see that Python and R are both *programming* languages and thus express algorithmic concepts in a similar way; yet spreadsheets, too, have been argued to constitute a programming language [11, 12] and were even used in the context of BX [13]. All three formats provide powerful mathematical and statistical features, which are either built in or provided by mature mathematical libraries [14]. The intended migration allows existing tools to remain in use while supported models are progressively transferred to a shared representation. The current prototype explores the transformations needed for this coexistence; adoption of the co-designed DSL is the next stage.



BENEVOL '26: The 25th Belgium-Netherlands Software Evolution Workshop 26-27 November 2026, Madrid, Spain
m.gerhold@utwente.nl (M. Gerhold); martijn.riemeijer@thermofisher.com (M. Riemeijer); o.f.sayilir@utwente.nl (Ö. Sayilir); vadim@grammarware.net (V. Zaytsev)
0000-0002-2655-9617 (M. Gerhold); 0009-0009-8860-2316 (Ö. Sayilir); 0000-0001-7764-4224 (V. Zaytsev)
© 2026 This work is licensed under a "CC BY 4.0" license.

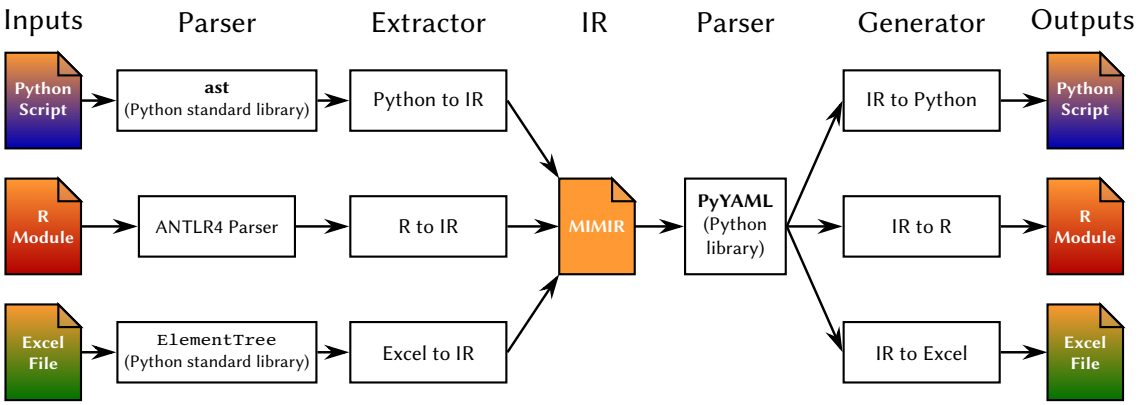


Figure 1: A detailed overview of the transformation pipelines that exist around the MIMIR format.

The implementation of our proof of concept was built using *expert vibing* [15], a methodology where we use our domain knowledge together with general knowledge about model transformations, parsing and intermediate representations, to rapidly prototype this project with the help of generative artificial intelligence (in our case Codex).

Following the priorities of Thermo Fisher Scientific, the prototype currently supports transforming programs that perform simple statistical simulations and programs that perform direct Monte Carlo simulations. A high-level overview of the multidirectional transformation is shown on the first page.

2. Technical overview

At the time of writing, MIMIR (Mathematical Interchange Model Intermediate Representation) supports three model formats, namely: Excel files, R modules, and Python scripts. All three can be used as transformation inputs and targets. Figure 1 illustrates the transformations that exist to and from the MIMIR format, which takes the form of a YAML file. MIMIR currently supports two input types: simple mathematical programs and direct Monte Carlo simulations.

The left section of the figure illustrates how the input files get processed and transformed into a MIMIR representation. Python and R source code can be parsed and processed directly, since these are stored as plain text. Excel files are archives that store worksheet data as Office Open XML (OOXML) [16] files, which need to be extracted first, after which the worksheets can be parsed as any regular XML file.

Since all transformation machinery was written in Python, we can use the Python and XML parsers shipped with its standard library to parse the contents of the Python and Excel inputs, respectively. In the case of R, the input is parsed using an R parser that was generated using the ANTLR4 parser generator [17], based on the R grammar from the ANTLR4 grammars repository [18]. After the input is parsed, it is passed to the appropriate extractor, which translates the input models to a MIMIR model.

The right section of the figure depicts how the IR is parsed, this time relying on the PyYAML library [19], and is transformed into one of the three models again. R and Python use an architecturally similar output pipeline, with only language specific deviations from each other. The pipelines first read what type of input the IR was based on, and then generate the corresponding output program, complete with imports of any libraries that are required to run the resulting program. The Excel generator also first starts off reading the IR to determine what to generate, but differs in how it creates the output. First, one or more OOXML files are built, according to what is needed for the output. For example, for cases that include visualisations, the files created have one sheet with the actual figures, one sheet with the input data, and one sheet with metadata. Then, these sheets are bundled to create a new `.xlsx` file, that can be opened and edited in spreadsheet software such as Microsoft Excel.

Input contracts When providing the initial input model for transformation, the output is determined by predefined *contracts* for each supported input type. For example, as illustrated in Listing 1, when

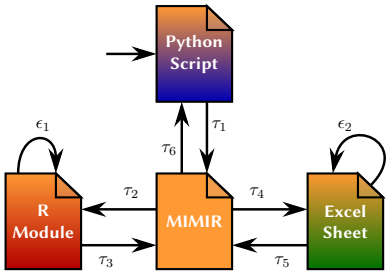


Figure 2: The chain of transformations performed to transform the profit calculator written in Python to R, Excel, and back to Python again through MIMIR. In this figure, τ transitions depict a transformation, and ϵ transitions depict a manual edit.

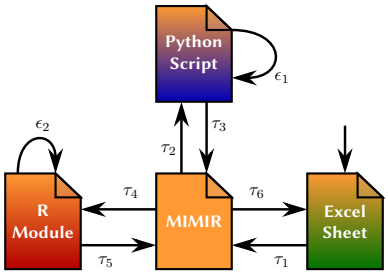


Figure 3: The chain of transformations performed to transform the Monte Carlo simulation created using an Excel template to Python, R, and back to Excel again through MIMIR. In this figure, τ transitions depict a transformation, and ϵ transitions depict a manual edit.

dealing with input that performs simple calculations, the R and Python contracts consist of first defining the values needed for the calculations, followed by a function performing them. Similarly for Excel, the expected input for this category of transformations is a worksheet that contains values and simple calculations, such as arithmetic and boolean tests, that are performed with them.

```python
# values to be used
units_sold = 1200
unit_price = 15.0

# function encoding the mathematical model
def evaluate_model(units_sold=units_sold,
    unit_price=unit_price):
    revenue = units_sold * unit_price
    return {"revenue": revenue}
```

```r
# values to be used
units_sold <- 1200
unit_price <- 15.0

# function encoding the mathematical model
evaluate_model <- function() {
  revenue <- units_sold * unit_price
  list(revenue = revenue)
}
```

Listing 1: Simple transformation inputs in Python and R.

2.1. The MIMIR intermediate representation

As mentioned before, MIMIR models are stored in a YAML format. We now go over the top-level concepts found in this format and explain what they contain:

The `program` section contains metadata about the source model and the target model, such as their file type and path. The `locations` section contains canonical concepts of the model, such as quantities, formulas, visualisation details, and random variables (as understood in probability theory). The `source_mappings` store how source model content, such as variable names, function names, and spreadsheet cells, maps to the defined `locations`. Similarly, the `target_mappings` store how target model content, such as variable names, function names, and spreadsheet cells, maps to the `locations` when rendering the output model. The `transforms` section contains an ordered history of transformations, while the `metadata` sections store descriptive, non-essential, information.

The contracts also play a role with change propagation using MIMIR. Users can change output models, as long as they still adhere to the contracts. The MIMIR mappings can then be updated using this modified output model, after which the updated IR can be used to generate a new version of the initial source model.

3. Illustrative examples

We now present two illustrative examples of how MIMIR can be used in practice. With these examples we particularly would like to emphasise how MIMIR facilitates collaboration among experts using different tools. These examples illustrate how calculations and subsequent changes can cross tool boundaries during such a transition.

3.1. Profit calculator

```python
UNITS_SOLD = 40
UNIT_PRICE = 12.50
UNIT_COST = 7.25
FIXED_FEE = 80.0

def evaluate_model(
    units_sold: int = UNITS_SOLD,
    unit_price: float = UNIT_PRICE,
    unit_cost: float = UNIT_COST,
    fixed_fee: float = FIXED_FEE,
) -> ProfitCalcResult:
    revenue = units_sold * unit_price
    variable_cost = units_sold * unit_cost
    contribution = revenue - variable_cost
    net_profit = contribution - fixed_fee
    return ProfitCalcResult(
        revenue=revenue,
        variable_cost=variable_cost,
        contribution=contribution,
        net_profit=net_profit,
    )
```

Listing 2: A simple profit calculator in Python.

```python
units_sold = 40
unit_price = 12.5
unit_cost = 7.25
fixed_fee = 80
revenue = (units_sold * unit_price)
variable_cost = (units_sold * unit_cost)
contribution = (revenue - variable_cost)
net_profit = (contribution - fixed_fee)
margin = (net_profit / revenue)
g48 = (net_profit > 100)

def evaluate_model():
    return {
        'revenue': revenue,
        'variable_cost': variable_cost,
        'contribution': contribution,
        'net_profit': net_profit,
        'margin': margin,
        'g48': g48,
    }
```

Listing 3: The profit calculator after the transformations and changes depicted in Figure 2.

Figure 2 depicts an illustrative example of how MIMIR can be used to work on the same mathematical model using different tools. We showcase this using a simple profit calculation, written by a Python expert, which is then transformed and extended by R and Excel experts. Listing 2 depicts the initial Python input for this transformation. In the first step this Python program is transformed to R (τ_1 depicting the step from Python to MIMIR and τ_2 depicting the step from MIMIR to R). Once the program is transformed to R, the R expert can extend the calculation by adding `margin <- net_profit / revenue` which adds a margin calculation to the output (depicted by ϵ_1). This change is then synced to the MIMIR model, which is depicted by τ_3. From the updated MIMIR model the Excel expert can generate a corresponding workbook (depicted by τ_4). The Excel expert then extends the model by adding a check if the net profit exceeds a certain threshold by adding the formula `=G47>100` in cell `G48`, with cell `G47`, automatically populated by MIMIR, containing the net profit. The Excel expert also updates the MIMIR model with their changes (depicted by τ_5). Finally, the Python expert can use the MIMIR model to generate a Python file (shown in Listing 3) with the changes of their colleagues present.

The shape of the resulting Python code is now simplified, the margin calculation is present, and so is the threshold check. Since no named range [**?**] is used for this cell, essentially making the variable nameless, MIMIR derives the threshold check's variable name from its original spreadsheet cell coordinate. We see that the regenerated program contains the added formulas, but its organisation and calling interface differ from the original (one accepts parameters and computes on invocation; another accepts nothing explicitly and stores its result on the module level; so the importer also must accept calculations at module scope). This example illustrates propagation of supported calculations and edits; preservation of the original program interface remains outside the demonstrated scope.

3.2. Direct Monte Carlo simulations

To showcase the potential feasibility of MIMIR in the industrial context, let us present a public illustrative example addressing a company-prioritised simulation category. Mathematical model templates that perform certain types of simulations are commonplace in industry, with the idea that experts only need to fill in the details of the calculations rather than building them from scratch each time. To simulate this scenario we have taken a publicly available Excel workbook that performs a direct Monte Carlo simulation [20], stripped it back to its basic shape, turning it into a template that can be reused for

these types of simulations. Using this template, we have built pipelines that transform instances of the template into equivalent Python and R simulations and back, and that can also convert simulations originally written in Python or R into instances of the template.

An example of a round trip transformation of this simulation is shown in Figure 3. Here, we start with an instance of the Excel template modelling a profitability simulation for bringing a new product on the market. We start by transforming this initial workbook to a Python script that performs an equivalent simulation, which is depicted by transitions τ_1 and τ_2 in Figure 3. This Python script can either be run directly or be used to modify the simulation, such as by changing probabilities and other values used for the calculations, or changing details of the accompanying histogram (depicted by transition ϵ_1). After transforming to R, the same features become available for this language as well (depicted by transitions τ_3, τ_4, and ϵ_2). Finally, when making the round trip complete and transforming back to Excel (transitions τ_5 and τ_6), the changes made in Python and R will be present in the resulting workbook.

With this round-trip transformation sequence, we have observed the current strengths and shortcomings of our prototype. The strategy of utilising templates has two advantages. The first advantage is that this enables the transformation architecture to detect easily what type of input it is dealing with, so it can generate the appropriate MIMIR model. The second advantage is that the generator can use the templates to create output models of the appropriate category in a well-formatted and readable manner. However, this strategy also turned out to be a weakness. The input contracts for these template-based models are much stricter than their simpler free-form counterparts. Changes made by users to these generated outputs must therefore observe the input contracts carefully, in order for future transformations to still work. Overall, we believe that this strategy is still worth using, especially in contexts where using templates to create new mathematical models is already commonplace.

4. Lessons learnt so far and ongoing work

The prototype exposes a tension between predictable transformations and the freedom to modify generated artefacts: templates simplify extraction and generation, but their input contracts constrain subsequent changes. This makes identifying the modelling idioms that the shared language must accommodate a central task for our ongoing co-design with domain experts.

After demonstrating this proof of concept, we are designing a scalable and robust version of MIMIR, fit for use in the industrial case provided by Thermo Fisher Scientific. For this new iteration, the intermediate representation takes on the shape of a domain specific language that is intended to make mathematical models easier to read and write than the current YAML representation. The syntax of the MIMIR language is co-designed with the end-users, which allows us to define a clear image of what does and does not fall within the desired scope of the format.

The industrial reality, which has no shortage of variety, demands a modular approach, where support for more languages and tools, alongside additional mathematical models, would need to be user-definable. We are designing MIMIR to expose a framework that can be used to expand the accepted source and target formats for transformations to and from MIMIR. Users can use this framework to map programming language or tool concepts to MIMIR concepts, which is used to guide the transformation between the formalisms. Another modularity enhancing design decision to make MIMIR an extensible solution is the addition of formula libraries. This addition allows users to create parametrised templates, which allows end-users to recycle scaffolding for frequently used formulae.

References

[1] K. Binder, D. W. Heermann, Monte Carlo Simulation in Statistical Physics: An Introduction, Graduate Texts in Physics, Springer International Publishing, 2019. doi:10.1007/978-3-030-10758-1.

[2] D. Frenkel, B. Smit, Understanding Molecular Simulation, Computational Science Series, 2 ed., Academic Press, San Diego, CA, 2001.

[3] P. Glasserman, Monte Carlo Methods in Financial Engineering, volume 53 of *Stochastic Modelling and Applied Probability*, Springer, 2003. doi:10.1007/978-0-387-21617-1.

[4] K. Beven, A. Binley, The Future of Distributed Models: Model Calibration and Uncertainty Prediction, Hydrological Processes 6 (1992) 279–298. doi:10.1002/hyp.3360060305.

[5] M. S. Arulampalam, S. Maskell, N. J. Gordon, T. Clapp, A Tutorial on Particle Filters for Online Nonlinear/non-Gaussian Bayesian Tracking, IEEE Transactions on Signal Processing 50 (2002) 174–188. doi:10.1109/78.978374.

[6] Microsoft Corporation, Microsoft Excel, 2026. URL: https://www.microsoft.com/en-us/microsoft-365/excel, spreadsheet software.

[7] Python Software Foundation, The Python Language Reference, Python Software Foundation, 2026. URL: https://docs.python.org/3/reference/, python 3 documentation.

[8] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2026. URL: https://www.R-project.org/.

[9] C. Moler, J. Little, A history of MATLAB, PACMPL 4 (2020). doi:10.1145/3386331.

[10] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A Fresh Approach to Numerical Computing, SIAM Review 59 (2017) 65–98. doi:10.1137/141000671.

[11] R. Pandita, C. Parnin, F. Hermans, E. Murphy-Hill, No Half-measures: A Study of Manual and Tool-Assisted End-User Programming Tasks in Excel, in: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2018, pp. 95–103. doi:10.1109/VLHCC.2018.8506540.

[12] E. Aivaloglou, D. Hoepelman, F. Hermans, Parsing Excel formulas: A Grammar and Its Application on 4 Large Datasets, Journal of Software: Evolution and Process 29 (2017) e1895. doi:10.1002/smr.1895, e1895 smr.1895.

[13] J. Cunha, J. Saraiva, J. Visser, From Spreadsheets to Relational Databases and Back, in: Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM), ACM, New York, NY, USA, 2009, p. 179–188. doi:10.1145/1480945.1480972.

[14] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array Programming with NumPy, Nature 585 (2020) 357–362. doi:10.1038/s41586-020-2649-2.

[15] V. Zaytsev, Compilation of Feature Models by Expert Vibing, in: Tool Transformation Contest (TTC), Joint Proceedings of the STAF 2025 Workshops, volume 4122 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: http://ceur-ws.org/Vol-4122/paper22.pdf.

[16] ECMA, ECMA-376: Office Open XML File Formats, https://ecma-international.org/publications-and-standards/standards/ecma-376/, 2021. 5th ed.

[17] T. Parr, The Definitive ANTLR 4 Reference, Pragmatic Bookshelf, 2013.

[18] T. Parr, R, Grammars-v4, https://github.com/antlr/grammars-v4/blob/master/r/R.g4, 2013.

[19] T. Müller, I. döt Net, M. Davis, PyYAML, https://pyyaml.org/, 2006–2020.

[20] M. Girvin, BI 348: Business Analytics: Data Analysis and Decision Making using Excel, Highline College course webpage, 2025. URL: https://people.highline.edu/mgirvin/AllClasses/348/348/BI348.htm, chapter 11, Video 66: Monte Carlo Simulation for New Product, 3 Uncertain Variables; includes the workbook BI348-Chapter11-Video66-NewProduct.xlsx which was used to create the template.