

Rapid Software Prototyping from Business Artifacts

Michael Wahler¹, Enrico Conte², Martin Frick¹, David Mosquera¹ and Marcela Ruiz¹

¹Zurich University of Applied Sciences (ZHAW), Winterthur, Switzerland

²Hitachi Energy, Baden, Switzerland

Abstract

Information systems must be able to quickly adapt to changing requirements. In software development, changing the requirements often entails a repetition of development tasks such as implementation or testing. To accelerate such tasks, *model-driven development* (MDD) offers users to express their requirements as domain models and to automatically generate code from these models. Most MDD approaches, however, require users to learn new tools and languages. Thus, MDD has found little adoption beyond a few niches and most code is still written by hand, which is slow, costly, and error prone.

In this paper, we propose an approach to using existing business artifacts as reusable models in software development. These models can be either transformed into executable code, or they can be used as functional modules using an interpreter. This approach allows product owners to update requirements and build information systems by having the behavior and logic of existing business artifacts immediately reflected in the software, which drastically increases business agility. We validate our approach with a proof of concept for automatically transforming existing business logic and domain data encoded in an Excel document into a web service and generating a web-based user interface for it.

Keywords

Agile Information Systems Engineering, Business Agility, Model-driven Development

1. Introduction

Businesses must be agile to cope with changes in their environment. This is especially true in the context of digitalization, which often accelerates both the frequency and the amplitude of change [1]. Businesses that have software at their core may possess sufficient knowledge to react quickly to such changes by having their software development life cycle partially automated, e.g., through DevOps. For other companies, e.g., in the electrical industry, who often use legacy approaches to software development and lack the necessary change culture, changes to the business requirements often entail a lengthy process until these changes are reflected in their digital products or services [1].


Model-driven development (MDD [2]) offers a solution for accelerating the path from requirements to implementation in software development. In MDD, domain experts (such as product managers) capture the requirements in structural and behavioral models. Software developers provide model transformations and code generators to automatically convert these models into executable code. At least, this is the theory. In practice, MDD is not widely used today in general-purpose software projects. While it has been successful in a few niches such as industrial automation [3] or digital electrical substations [4], we have observed that both

Agil-ISE 2022: Intl. Workshop on Agile Methods for Information Systems Engineering, June 06, 2022, Leuven, Belgium

✉ wahl@zhaw.ch (M. Wahler); enrico.conte@hitachienergy.com (E. Conte)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

product managers and software developers to often be reluctant to learn and use MDD tools for specifying applications. As a result, most code is still written by hand.

The key to success of the niches above is considered to be that they use *domain-specific languages* [5] over *general-purpose languages* (as, for example, provided by UML). This allows domain experts to express requirements in a formal language with which they can quickly get familiar. Thus, we see an opportunity for MDD if the semi-formal and formal approaches *that are already in use* to encode business artifacts can be adopted and integrated into a model-driven software engineering (MDSE) life cycle.

We propose an approach to using existing business artifacts as models in an MDSE process. This allows domain experts to apply changes to the requirements directly to their business artifacts and automatically generate a new release of the software. Through such automatic integration of artifacts from a domain-specific language (e.g., spreadsheets in Excel or illustrations in Visio) into the software development life cycle, companies can rapidly develop applications and thus, react to changing business requirements in an agile way. In this paper, we address the **research question**: *Can business artifacts such as spreadsheets be used to automatically generate software prototypes?*

1.1. Related Work

Office documents such as Excel spreadsheets can be considered *domain-specific languages* [6], which allow domain experts to encode problems in their respective domain with a familiar terminology. Using well-known languages and applications (such as spreadsheets) in modeling solves some of the “grand challenges” in model-driven development suggested by A. Bucchiarone et al. [7] because they eliminate the need for domain experts to learn new modeling languages or to get familiar with a new tool.

Although office documents are widely used in many domains and even have inspired some approaches to automating the software development life cycle such as low-code platforms [5], they are usually only used as second-class artifacts in software development. As an example, they are used to specify domain data and business logic (as a starting point for software development), or, reversely, to represent the call graph of the functions in the execution of a program [8].

Excel documents have been used in an approach to automatically generating web pages [9], which focuses on the visual aspects of representing a spreadsheet as a web page. Although this approach is useful, it is unclear how formulas in the spreadsheets are calculated.

Several others have started to see the value of making the data and calculations in spreadsheets available through an API. As an example, Microsoft Graph [10] provides a REST API to Excel documents stored in its cloud to allow for their integration in web applications. There are open-source solutions such as the GitHub project excel-microservice [11], which provides an API for manipulating Excel documents through a microservice API. There are even commercial solutions available for converting Excel documents into applications such as SpreadsheetWeb [12].

2. Motivating Example

In this section, we present an example application from the domain of electrical substations, which are nodes in electrical grids. They are composed of many different types of electrical

assets, e.g., circuit breakers or transformers. Substation owners must ensure their optimal operation at all times. This can be achieved with a balanced mix of maintenance, replacement of aging assets, and availability of spare parts. A key element in this optimization problem is the storage of “enough” spare parts.

At Hitachi Energy we have extended an existing algorithm for the calculation of the optimal amount of spare parts to be kept in storage, which will ensure the best balance between cost of spares and risk of power interruptions. We have built a proof-of-concept model for this algorithm using Microsoft Excel. Two steps must be taken to further transform this proof of concept into a prototype: first, validation with product managers, and second, scale-up to represent a real-world problem.

The challenge with the first step is that complex Excel models can be hard to understand by anyone else than their creator. Thus, we seek a method to quickly convert such models into a tool that can be handled, tried, and validated by others. We want to hide the intricacies of the models and highlight the input/outputs in a UI wrapper, without the effort and the know-how for such development.

The challenge with the second step is that an excel model can handle a handful of variables, but it becomes impractical or impossible to use when the number of variables increases to represent a real scale problem, e.g., with hundreds or thousands of spare parts. In this study we focus on the first challenge, but a natural continuation will be to explore how to automatically extract the logic of the algorithm and implement it into a software, so that it can be scaled up at wish. Figure 1 shows several of the input parameters to the optimization algorithm in Excel.

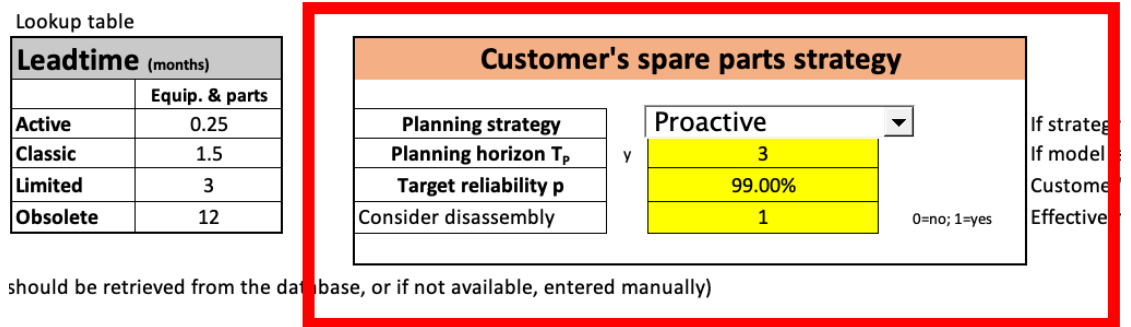


Figure 1: Parts of the business logic encoded in Excel

3. Proposed Implementation

We propose the model-view-controller (MVC) design pattern [13] for the system architecture, which splits the responsibilities of an application into the *model*, which contains the data and business logic; the *view*, which interacts with the user through an interface; and the *controller*, which takes input from the view, manipulates the model, and updates the view.

We use a *model interpretation* approach [6] for turning Excel documents into executable artifacts. In model interpretation, an engine is built that wraps around the model (i.e., the Excel document) and reads or modifies it on request. For preparing an Excel document to be used

with this approach, a domain expert must declare one or more cells as inputs to the calculations in the document, and one or more cells as outputs of these calculations.

In this section, we show our proof-of-concept implementation in Python to demonstrate how we have used the MVC pattern to automatically generate a web application from an Excel document.

The Model The model is the Excel document that contains both business logic and (often) domain data. Instead of generating code from the model (which is the most common approach in MDD), we follow the model *interpretation* (or *models at runtime*) approach: the Excel document remains unchanged, and we add a wrapper around it to enable programmatic access.

This offers several advantages with respect to business agility. First, changes to the model are immediately reflected in the application because no code needs to be generated. Second, the application does not have to be stopped when the business logic is updated, which allows for dynamic software updates [14]. Third, fewer bugs will be introduced with each change to the model because there is no additional hand-written code.

For our proof-of-concept implementation, we have developed a wrapper that provides a REST API to change cells, trigger calculations, and read cells of an Excel document. This REST API allows the *view* and the *controller* to interact with the Excel sheet. To this end, we use the Python package `formulas`¹ to update data in Excel sheets and trigger the calculation of formulas in the sheets and the package `flask`² to build a web API for the Excel document.

This results in a microservice, which can be immediately integrated into existing web applications or serve as the foundation of new applications. As an example, the API call

```
curl -H "Content-Type: application/json" -d '{"ExcelFileName": "simple.xlsx",
      "a": "5"}' -X POST http://localhost:5000/ExcelCalculator
```

updates the input cell named “a” in the Excel sheet “simple.xlsx” to the value 5. Then, the calculations in that sheet are triggered and the results of the calculations are returned in the JSON output `{ "result": [6.0, "text"] }`.

Besides updating values in the Excel document and recalculating the formulas contained therein, our web service provides an API to add rows to the tables in the document. This enables users to add additional data to the document at runtime and calculate functions that operate on a set of values (such as *sum* or *average*).

The View The view is realized through a web page, which is automatically generated from the data in the model. In our implementation, we use the well-established stack of HTML, CSS, and JavaScript for the front-end of the view and Python (again, the `flask` package) for its backend. Our approach provides a URL `/webform/<filename>`, which automatically renders a `flask` HTML template. An example view that was automatically generated is shown in Figure 2. In this figure, the relation to the elements in the Excel document (cf. Figure 1) are obvious: parameters of the spare parts optimization algorithm such as the planning strategy and the planning horizon have become input elements on the web page. The results of the optimization

¹<https://pypi.org/project/formulas/>

²<https://flask.palletsprojects.com/en/2.0.x/>

algorithm become outputs on the web page as simple text fields or graphical widgets. In our implementation, we use gauges and column charts from Google Charts [15] to display widgets.

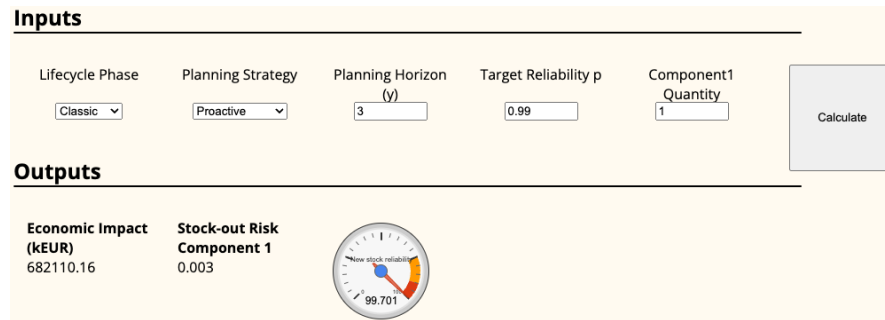


Figure 2: Automatically generated view for the spare parts model

The Controller The controller is a set of JavaScript functions that send the value of the input fields from the web page (the *view*), sends it to the backend (the *model*), and updates the view with the results. These JavaScript functions are parameterized in the automatic generation of the view. They are called when the *Submit* button is clicked on the web page and interact with the model in the backend.

4. Conclusion and Future Work

The answer to our research question “*Can business artifacts such as spreadsheets be used to automatically generate software prototypes?*” is a clear *yes*: we have presented an approach to rapid software prototyping by using office documents as first-class modeling artifacts and integrating them into existing applications through code generation and model interpretation. Any changes in the office documents are immediately reflected in the application without further design or implementation effort, which significantly increases agility in information systems engineering.

The presented approach provides an architecture and proof-of-concept implementation with several limitations and open questions, which we enumerate for further discussion in the workshop.

1. We have shown that spreadsheets can be valuable artifacts for rapid software prototyping. We are going to explore systematically what other artifacts (e.g., technical drawings, Gantt charts) can be automatically integrated into the software development process.
2. The manual mapping of inputs and outputs in the Excel document could be automated by identifying all existing formulas and their input cells in the Excel document and assigning them as outputs in the `Outputs` and `Inputs` sheets.
3. The Python package that we use to access Excel documents does yet not support all function types of Excel. There are alternative APIs available for Python and other languages that should be investigated in terms of supported Excel features and functions.

4. We have not yet sufficiently measured the performance of our approach. First experiments with measuring the response time of the generated web service resulted in 30-90 ms for a simple calculation with a few inputs and one formula.
5. We want to explore how our approach can be used to build a test oracle to validate software that is developed with the Excel document as specification.

References

- [1] N. Perkin, P. Abraham, *Building the agile business through digital transformation*, Kogan Page Publishers, 2021.
- [2] B. Selic, The pragmatics of model-driven development, *IEEE software* 20 (2003) 19–25.
- [3] IEC, ISO/IEC 61131-3:2013: Programmable Controllers - Part 3: Programming Languages, 2013.
- [4] IEC 61850 Communication Networks and Systems for Power Utility Automation, IEC, 2003.
- [5] A. Bucchiarone, F. Ciccozzi, L. Lambers, A. Pierantonio, M. Tichy, M. Tisi, A. Wortmann, V. Zaytsev, What is the future of modeling?, *IEEE software* 38 (2021) 119–127.
- [6] M. Brambilla, J. Cabot, M. Wimmer, Model-driven software engineering in practice, *Synthesis lectures on software engineering* 3 (2017) 1–207.
- [7] A. Bucchiarone, J. Cabot, R. F. Paige, A. Pierantonio, Grand challenges in model-driven engineering: an analysis of the state of the research, *Software and Systems Modeling* 19 (2020) 5–13.
- [8] J. Bézivin, Model driven engineering: An emerging technical space, in: *International Summer School on Generative and Transformational Techniques in Software Engineering*, Springer, 2005, pp. 36–64.
- [9] X. She, Y. Zheng, An automatic page code generation method based on excel template and poi technology, in: *2020 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, IEEE, 2020, pp. 560–564.
- [10] Microsoft, *Overview of Microsoft Graph*, <https://docs.microsoft.com/en-us/graph/overview>, 2022. Online; accessed 2022-01-18.
- [11] Weaver, *Excel Microservice*, <https://github.com/sysunite/excel-microservice>, 2022. Online; accessed 2022-02-07.
- [12] Pagos, *SpreadsheetWeb*, <https://www.spreadsheetweb.com/>, 2022. Online; accessed 2022-02-07.
- [13] M. Fowler, D. Rice, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.
- [14] M. Wahler, M. Oriol, Disruption-free software updates in automation systems, in: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 2014, pp. 1–8. doi:10.1109/ETFA.2014.7005075.
- [15] Google, *Google Charts*, <https://developers.google.com/chart>, 2022. Online; accessed 2022-02-07.