Spreadsheets are Models Too [Position Statement]

Richard F. Paige, Dimitrios S. Kolovos and Nicholas Matragkas
Dept. of Computer Science
University of York, UK
[richard.paige, dimitris.kolovos, nicholas.matragkas]@york.ac.uk

ABSTRACT

Spreadsheets are among the most widely used tools in systems engineering, especially for documenting system requirements and tests, and for supporting tasks like impact analysis, traceability management and project planning. We argue for the treatment of spreadsheets as models, in order to support the systems engineering lifecycle, and to provide a suitable migration path for organisations to follow in maturing their use of modelling techniques.

1. INTRODUCTION

In Model-Driven Engineering (MDE) approaches to systems engineering, many different languages are used (e.g., UML, SysML, domain-specific languages). Usually such languages are designed and implemented by MDE specialists, who use metamodelling infrastructure (e.g., EMF/Ecore[1]) to define the abstract syntax of such languages, and thereafter exploit the infrastructure for the purposes of automation – for example, generating code or text, version control, validation, etc. Once languages with metamodels have been provided, automated model management tools and techniques can be used for systematically manipulating and modifying models across the engineering lifecycle. In particular, tools such as Obeo Designer[2], Epsilon [3], or ATL [1] can be applied to support different engineering tasks.

Systems engineering is expensive and complex, often involves multiple engineering disciplines (e.g., in avionics or aerospace, it can involve software, mechanical, materials and power engineering), and substantial communications overhead between skilled personnel with different vocabularies, practices and tools. Arguably, MDE as it is currently practiced (and supported by tools) is insufficient for supporting the full systems engineering lifecycle. In particular, it can very easily fall short in the early stages, when requirements are still being elicited. Often, early requirements are ambiguous and need to be described in unconstrained natural language: using a domain-specific language may place too many constraints (both conceptual or structural) on specification. As well, requirements often emerge from previous developments, and these requirements may have been specified in non-MDE languages. Combine this with the gradual increase in MDE skills, there is substantial benefit to be able to *interface* MDE languages and tools with non-MDE languages and tools.

In this position paper, we argue for interfacing spreadsheets with MDE languages and tools. We provide motivation for doing this, and briefly touch on some the important technical challenges of, and alternatives for doing so.

2. MOTIVATION

There have been a number of contributions made related to integrating spreadsheets into an engineering process. Much of this work focuses on using software engineering practices to improve the quality of spreadsheets. This includes work on bad smell detection and visualisation in spreadsheets [6], and other analytic approaches that exploit assertions to identify formula errors [8], or that provide testing techniques for spreadsheets [7]. Constructive approaches such as [4, 2] focus on generating high quality spreadsheets using transformation approaches. None of this research has taken the perspective of treating spreadsheets as models.

We have hinted at a number of motivations for treating spreadsheets as models, and for supporting the use of model management operations (such as model transformations) on spreadsheets. We briefly summarise key motivations.

- *Early stages of engineering.* MDE operates most efficiently on well-defined languages (that do not change frequently, or at least, not in significant ways) and models with limited uncertainty. In the early stages of requirements engineering, the concepts of interest in our models may change frequently; they may be imprecisely defined; and the languages that we use to express these concepts may need to evolve. MDE techniques may not be the most useful or appropriate in early stages. Natural language with some restrictions is widely used for early requirements engineering, as are tables of natural language requirements. These can easily be expressed using spreadsheets, which also enable traceability and (in later stages) requirements coverage analysis. Being able to treat spreadsheets

[1] http://www.eclipse.org/emf
[2] http://www.obeodesigner.com/download

as models thus enables defining bridges between early stages of systems engineering, and later stages, where more precise languages are needed.

- *Support for legacy models.* Industry uses spreadsheets, and many large organisations have legacy spreadsheets that can play critical roles, such as in project configuration and monitoring/measurement, requirements capture for product lines, etc. Being able to use such legacy spreadsheets as-is with new engineering processes, practices and tools makes it easier to change processes and practices while reducing risk of bad effects on the bottom line.

- *Tabular problems need tabular solutions.* Some modelling problems are inherently tabular in nature, and benefit from being able to specify data (models) in columns and rows (with constraints amongst them) without requiring relational solutions. Specification of control laws, or parameters used to configure product lines, simple requirements capture, and test suite specification are all problems that lend themselves to tabular specifications, where spreadsheets can conceivably provide support. Providing MDE support for such idioms allows engineers who need to use such concepts to benefit from automated processing support.

- *Supporting existing skillsets.* Not every organisation has, or can quickly acquire, expertise in MDE and model management. Most organisations do have expertise and skills with spreadsheets. Providing means for organisations to transition gradually to use of MDE and model management, and allowing those organisations to maximise the use of their current skillset, could reduce the risks associated with adopting MDE.

- *Catching repeated errors.* Substantial research has been carried out in MDE in terms of automated support for identifying and repairing repeated errors in modelling and model management. For example, updating models or evolving models after changes in a modelling language are problems for which good automated or semi-automated solutions exist. These are problems with spreadsheets as well (e.g., bad smell detection). By interfacing spreadsheets with MDE, it may be that spreadsheet users can exploit MDE solutions.

3. MECHANISMS

There are several plausible ways to interface spreadsheets and MDE.

- Build *injectors* which generate models (with metamodels) from spreadsheets, thus allowing MDE languages and tools to be applied to spreadsheets indirectly. Additionally, extractors from models to spreadsheets may also be needed in order to return results to a form amenable to processing by spreadsheet tools. In both the injection and extraction, *specification blow-up* may be an issue (i.e., encoding or decoding spreadsheets as or from models may lead to less than optimal spreadsheet or model sizes or structures).

- Provide equivalents of MDE and model management operations on spreadsheets, e.g., update-in-place transformations, validation/constraint checking, transformations, text generation. These would need to be encoded using any scripting languages provided by a spreadsheet tool. For example, for Google Spreadsheets, these operations might be encoded using the Spreadsheet Service[3]. However, such encodings would need to be reimplemented for each spreadsheet tool.

- Provide spreadsheet drivers for model management tools, so that these tools can directly manipulate spreadsheets like any other form of models. This is the approach we have taken in Epsilon [5]. A driver must be implemented for each spreadsheet tool - though some abstraction is possible (specifically, a spreadsheet interface is provided that needs to be implemented for each spreadsheet tool). Arguably, implementing an interface for querying and changing spreadsheets via an API is less expensive than implementing model management operations for each spreadsheet tool.

4. CONCLUSIONS

Spreadsheets are models: a less constrained and less expressive form of model than those permitted by full-blown MDE languages and tools. By treating spreadsheets as models, we can provide ways to bootstrap the MDE process, to enable automated and powerful tool support for legacy models, and a way to maximise use of current skillsets while personnel are educated in using MDE and model management techniques. Arguably, MDE and model management tools should support more model/data representation formats and techniques like spreadsheets, which allow more flexible and less constrained styles of specification and design.

5. REFERENCES

[1] Atlas Transformation Language, official web-site. http://www.sciences.univ-nantes.fr/lina/atl/.

[2] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva. MDSheet: A framework for model-driven spreadsheet engineering. In *Proc. ICSE*, 2012.

[3] Dimitrios S. Kolovos, Louis M. Rose, Antonio Garcia Dominguez and Richard F. Paige. *The Epsilon Book*. 2013. http://www.eclipse.org/epsilon/doc/book/.

[4] G. Engels and M. Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proc. ASE'05*, ASE '05. ACM, 2005.

[5] M. Francis, D. S. Kolovos, N. Matragkas, and R. F. Paige. Adding spreadsheets to the MDE toolkit. In *Proc. MoDELS*. LNCS 8107, Springer-Verlag, 2013.

[6] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *ICSE*, pages 441–451. IEEE, 2012.

[7] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. *ACM Trans. Softw. Eng. Methodol.*, 10(1):110–147, Jan. 2001.

[8] J. Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal of Visual Languages & Computing*, 11(1):49 – 82, 2000.

[3] `https://developers.google.com/apps-script/reference/spreadsheet/`