Coupled Evolution of Model-Driven Spreadsheets

Jorge Mendes
HASLab / INESC TEC, Universidade do Minho, Portugal
jorgemendes@di.uminho.pt

***Abstract*—Spreadsheets are increasingly used as programming languages, in the construction of large and complex systems. The fact is that spreadsheets, being a highly flexible framework, lack important programming language features such as abstraction or encapsulation. This flexibility, however, comes with a price: spreadsheets are populated with significant amounts of errors.**

One of the approaches that try to overcome this problem advocates the use of model-driven spreadsheet development: a spreadsheet model is defined, from which a concrete spreadsheet is generated. Although this approach has been proved effective in other contexts, still it needs to accommodate for future evolution of both the model and its instance, so that they remain synchronized at all moments.

In this paper, we propose a pair of transformation sets, one working at the model level and the other at the instance level, such that each transformation in one set is related to a transformation in the other set. With our approach, we ensure model/data compliance while allowing for model and data evolution.

***Keywords*-Spreadsheets; Model-Driven Engineering (MDE); Software Evolution**

I. Introduction

Spreadsheets are widely used by non-professional programmers, the so-called *end users*, to develop business applications. Spreadsheet systems offer a high level of flexibility, making it easier to get started working with them. This freedom, however, comes with a price: spreadsheets are very error prone, as shown by numerous studies which report that up to 90% of real-world spreadsheets contain errors [1].

In recent years the spreadsheet research community has recognized the need to support end-user *model-driven spreadsheet development* (MDSD), and to provide spreadsheet developers and end users with methodologies, techniques and the necessary tool support to improve their productivity. Along these lines, several techniques have been proposed, namely the creation of spreadsheet templates [2], *ClassSheets* [3] and the use of class diagrams to specify spreadsheets [4]. These proposals guarantee that end users safely edit their spreadsheet data. In fact, they introduce a form of model-driven software development: a spreadsheet business model is defined from which a customized spreadsheet application is generated guaranteeing the consistency of the spreadsheet data with the underlying model.

In the past, we have proposed two techniques to provide end users with an environment for MDSD: the embedding of *ClassSheet* models in a spreadsheet system [5], and the co-evolution of *ClassSheets* and spreadsheet instances [5], [6]. With our work, it is possible to automatically obtain an updated spreadsheet instance given that some evolution is performed at the model level. This approach, however, does not consider the evolution of the spreadsheet instance and the necessary co-evolution of the corresponding model. Nevertheless, this is direction also worth studying since, for example, several operations are easier to realize on the instance level.

In this paper we propose the integration of bidirectional [7] and coupled transformation [8] techniques in our formal co-evolution setting, so that we enable the evolution of spreadsheet instances and the automatic co-evolution of spreadsheet models.

II. ClassSheet Models

ClassSheets [3] are a high-level, object-oriented formalism to specify the business logic of spreadsheets. *ClassSheets* allow users to express business object structures within a spreadsheet using concepts from the Unified Modeling Language (UML). Using the *ClassSheets* model, it is possible to define spreadsheet tables and to give them names, to define labels for the table's columns, to specify the types of the values such columns may contain and also the way the table expands (*e.g.*, horizontally or vertically).

Besides a textual (and formal) definition, *ClassSheets* also have a visual representation which very much resembles spreadsheets themselves [9]. We have embedded such visual model representation that mimics the well-known embedding of a domain specific language in a general purpose one [5]. Like in such embeddings, we inherit all the powerful features of the host language: in our case, the powerful interactive interface offered by the (host) spreadsheet system.

III. Coupled Evolution

To provide a synchronization between models and data, we consider the approach of coupled transformations where the consistency relation is the model compliance relation [8]. The diagram in Figure 1 shows the principle of such approach: starting from a model and one of its compliant data instances we apply a set of transformations on each artifact to obtain a new model and a compliant new instance. The goal is to make it possible to compose these transformations to provide a linear flow of model/data evolution.

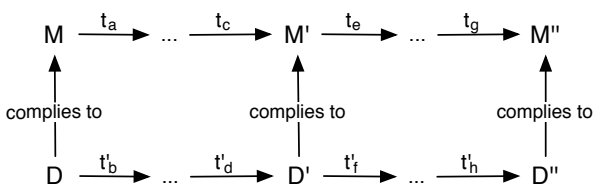

Figure 1. Representation of a set of transformations performed both on the model and the data.

We developed coupled transformations for model-driven spreadsheet evolution: the model is our *ClassSheet* model and the data is the spreadsheet data.

Figure 2 abstractly illustrates a *ClassSheet* model with three columns (left) and a spreadsheet instance with three repetitions of those columns (right). The *ClassSheet* model expands horizontally as suggested by the ellipsis.

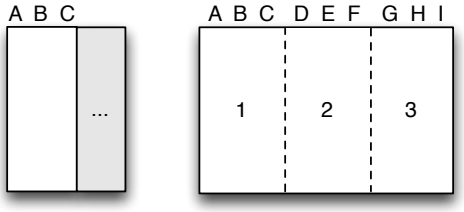

Figure 2. Example of an expandable row class model (left) and an instance (right).

A common spreadsheet evolution step is the addition of a column. In an MDE setting this can be done in two ways. The first corresponds to the column being added to the model and the data being co-evolved: if in the model of Figure 2 we add a fourth column, then all three class instances in the data need to be updated with a new column. In this case, an evolution in the model corresponds to several updates on the data. The second, and more challenging, evolution occurs when the user adds a column to the spreadsheet data. Figure 3 presents an example where the user added a column to the last class instance (column ◊ is added between columns H and I). In this case several models could be defined complying to the new data. We have used our own spreadsheet expertise to select one such model, the one we would manually create, but this is an approach that requires further validation.

We have implemented all evolution step functions in an algebraic and point-free setting, where each evolution function at the data (model) level is coupled with a co-evolution function at the model (data) level. Next, as a preview of our implementation, we show the signature of the function that adds a column to the spreadsheet data instance:

```
addColumData:(Model,Data)->Int->(Model,Data)
```

This function receives both the initial model and its instance and starts by adding the column to the instance in the

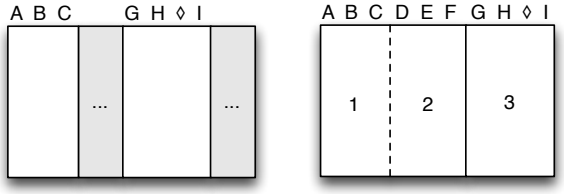

Figure 3. Example of two expandable row classes model (left) and an instance (right) after adding a column after column H in the data.

position given by the second argument; then, it automatically evolves the model so that it complies to the evolved instance. As a result, the function returns a pair with both the new model and the new spreadsheet data with one column added.

IV. Conclusions

We have been working on model-driven approaches for spreadsheets at both the theoretical [5], [10] and the practical [6] level. Moreover, we have integrated our techniques in a unifying framework for model-driven spreadsheet development, MDSheet [11]. In this paper we propose a model-driven approach to spreadsheet engineering. We studied one complex and relevant problem in model-driven engineering: the evolution of the model (data) and the automatic synchronization of the data (model). Thus, we propose a technique for the coupled evolution of both software artifacts. In fact, operations that are easier for users to accomplish on the model side can be performed there and, on the other hand, operations that are more suitable to be accomplished on the data instances can straightforwardly be achieved on data sheets.

In the lines proposed in this paper, another result has already been achieved [12]. In fact, we have proposed a prototype framework for the bidirectional development of spreadsheets and their models. Although this approach seems to validate the intentions envisioned in this paper, there are still several directions for future research. Firstly, we plan to consult spreadsheet experts, both from academy and industry, in order to spot opportunities for improvement. Secondly, we will conduct an empirical study with human spreadsheet users in order to try to assess the practical interest of our framework. Finally, we plan on using the insights learned in the context of the previous steps in the improvement of our framework, that we wish to transform into a fully-functional bidirectional system with a practical impact.

Acknowledgement

This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project ref. PTDC/EIA-CCO/108613/2008, and FCT grant BI4-2011 PTDC/EIA-CCO/108613/2008.

REFERENCES

[1] R. Panko, "Spreadsheet errors: What we know. what we think we can do." *Proceedings of the Spreadsheet Risk Symposium, European Spreadsheet Risks Interest Group (EuSpRIG)*, 2000.

[2] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert, "Visual specifications of correct spreadsheets," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, 2005, pp. 189–196.

[3] G. Engels and M. Erwig, "ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications," in *20th IEEE/ACM Int. Conf. on Automated Sof. Eng., Long Beach, USA*. ACM, 2005, pp. 124–133.

[4] F. Hermans, M. Pinzger, and A. van Deursen, "Automatically extracting class diagrams from spreadsheets," in *Proc. of the 24th European Conference on Object-Oriented Programming*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 52–75.

[5] J. Cunha, J. Mendes, J. P. Fernandes, and J. Saraiva, "Embedding and evolution of spreadsheet models in spreadsheet systems," in *IEEE Symp. on Visual Languages and Human-Centric Computing*. IEEE CS, 2011, pp. 179–186.

[6] J. Mendes, "Classsheet-driven spreadsheet environments," in *IEEE Symp. on Visual Languages and Human-Centric Computing*. IEEE CS, 2011, pp. 235–236.

[7] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger, "Bidirectional transformations: A cross-discipline perspective," in *Theory and Practice of Model Transformations, Second International Conference, ICMT 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings*, ser. Lecture Notes in Computer Science, R. F. Paige, Ed., vol. 5563. Springer, 2009, pp. 260–283.

[8] R. Lämmel, "Coupled Software Transformations (Extended Abstract)," in *First International Workshop on Software Evolution Transformations*, Nov. 2004.

[9] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger, "Automatic generation and maintenance of correct spreadsheets," in *Proc. of the 27th Int. Conf. on Software Eng.* New York, NY, USA: ACM, 2005, pp. 136–145.

[10] J. Cunha, J. Visser, T. Alves, and J. Saraiva, "Type-safe evolution of spreadsheets," in *Int. Conf. on Fundamental Approaches to Software Engineering*, ser. FASE'11/ETAPS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 186–201.

[11] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "MDSheet: A framework for model-driven spreadsheet engineering," in *34th International Conference on Software Engineering*. ACM, 2012, pp. 1412–1415.

[12] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva, "Bidirectional Transformation of Model-Driven Spreadsheets," in *5th International Conference on Model Transformation*, ser. LNCS, vol. 7307. Springer, 2012, pp. 105–120.