# ClassSheet-driven Spreadsheet Environments

Jorge Mendes
Departamento de Informática, Universidade do Minho, Braga, Portugal
jorgecunhamendes@gmail.com

## I. Introduction

Spreadsheet systems are well known and widely used in all kinds of business applications. They are used by tens of millions of people who create hundreds of millions of spreadsheets every day [1]. Spreadsheet systems are easy to use and very intuitive, allowing the aggregation of all types of data.

As software systems, however, spreadsheets are usually created by single end-users, without planning ahead of time for maintainability or scalability. Indeed, after their initial creation, many spreadsheets turn out to be used for storing and processing increasing amounts of data and for supporting increasing numbers of users over long periods of time.

Also, like any other software artifact, spreadsheets tend to evolve into large software systems where requirements, development and deployment platforms change. Spreadsheet systems, however, lack the support for evolution that one finds in other software systems!

With our work, we specifically target the problem of spreadsheet evolution.

In a recent paper [2], we have proposed to embed *ClassSheet* spreadsheet models [3] in spreadsheet systems themselves. In this approach a spreadsheet business model is defined from which a customized spreadsheet application is generated guarantying the consistency of the spreadsheet with the underlying model. Our approach closes the gap between creating and using a domain specific language for spreadsheet models and a totally different framework for actually editing spreadsheet data. Instead, we unify these operations within spreadsheets: in one sheet we define the underlying model while another sheet holds the actual data, such that the model and the data are kept synchronized by our framework.

An important feature of our approach is that model evolution is available as a set of pre-defined operations. Also, any evolution in a spreadsheet model is automatically propagated to its spreadsheet instance (that contains the underlying data). Our framework is such that editing spreadsheet data is also safe and controlled.

## II. Embedded *ClassSheets*

*ClassSheets* are a high-level, object-oriented formalism to specify the business logic of spreadsheets. They 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 [4]. Thus, such visual model representation makes developing the spreadsheet model very similar to creating a concrete spreadsheet. In order to support this visual representation, a specific interactive tool has been developed [5]. This tool provides a powerful interactive environment to create *ClassSheets* and to automatically generate spreadsheets that follow the specified business model. The generated spreadsheets guide end users in introducing data that follows the underlying model, thus avoiding several common spreadsheet errors.

We have embedded *ClassSheet* models in spreadsheet systems. In this embedding we mimic the well-known embedding of a domain specific language in a general purpose one. 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. This approach has two key advantages: first, we do not have to build and maintain a complex interactive tool. Second, we provide *ClassSheet* model developers the programming environment they are used to: a spreadsheet environment. Furthermore, because the *ClassSheet* model and the spreadsheet data are defined in the same environment, we now have the power to ensure that they are synchronized.

An example of a model specifying a system to mark student's grades, along with an instance for it is shown in Figure 1.

The top-left part of Figure 1 illustrates a sheet with the model, containing two tables: the first between rows 1 and 5 represents students' marks in exams and another between rows 7 and 10 representing students. Buttons on the top part of the spreadsheet allow the user to evolve the model (*e.g.* `Col+` introduces a new column).

The instance sheet is shown in the bottom-right part of the figure. In the instance it is possible to add exams and students only in the right places using the corresponding buttons.

## III. Spreadsheet Evolution

As widely recognized by the Model-driven Engineering (MDE) community, the manual co-evolution of models and instances is an error-prone task leading to inconsistencies between models and related artifacts. We created a spreadsheet environment where it is guaranteed that spreadsheet data always conforms to the evolved *ClassSheet* model.

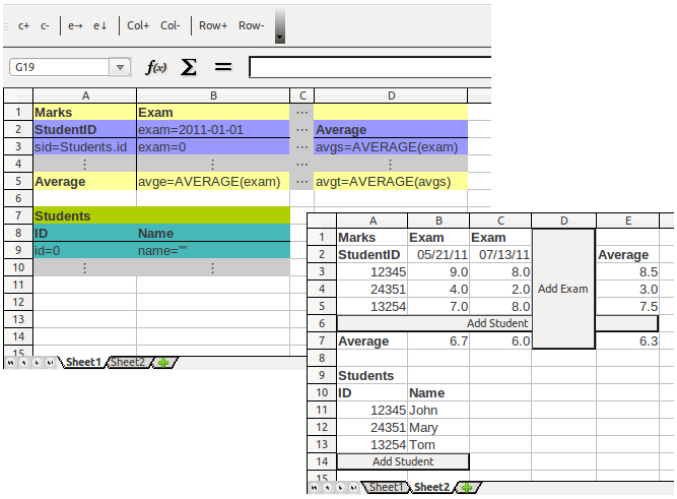


Fig. 1: Spreadsheet model and instance for a marking system.

In our setting, the *ClassSheet* evolution steps automatically produce (backward and forward transformation) functions to map the spreadsheet instance so that it conforms to the evolved model. The end user does not have to concern himself to adapt the data after a model evolution step: the generated forward and backward functions do this work for him! Our approach is illustrated in Figure 2.

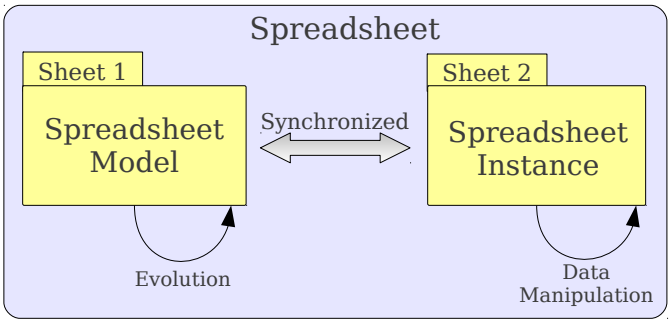


Fig. 2: Spreadsheet model/instance evolution scenario.

In fact, we have developed a prototype extension to a widely used spreadsheet system that allows the user to generate a spreadsheet skeleton from the sheet containing the model. This template is generated in a different sheet which is built with the necessary mechanisms to avoid that user commits errors: it contains several restrictions that always ensure the data conveys the model that abstracts it. Moreover, in the sheet where the model is defined we have created several buttons that allow the user to evolve the model (*e.g.* add a new column). These modifications are done both in the model's and in the data's sheets at the same time, so that both are always kept synchronized.

## IV. Future Work

In the past, we have shown that, under certain conditions, spreadsheet models other that *ClassSheets* can help spreadsheet end users being more productive [6]. Later, in [2], we have proposed that a *ClassSheet* spreadsheet model and the concrete spreadsheet it models should be defined in the same environment.

Although evidences suggest that our embedded approach is both practical and effective, we intend in the future to assess this hypothesis with an empirical study with end users. In these lines, we will organize and run an empirical study to evaluate: *a)* how spreadsheet users (end users or/and professional users) adapt to our embedded discipline of spreadsheet development and *b)* what is the improvement of the embedded system on spreadsheet users productivity, both in terms of efficiency and effectiveness.

Moreover, we will study new bidirectional/co-evolution techniques that can automatically synchronize models and instances after an evolution step in an instance. So far, our data-refinement approach can only synchronize the instances after a model change and not the other way around. We guarantee model/instance synchronization by restricting the editing possibilities on the instance side: while general edition steps are allowed on the model side, the user can not, for example, add a new column to an instance. This means that if a spreadsheet user would be allowed to update an instance by one such addition the available techniques would not able of generating the new *ClassSheet* model for it (although, we may still obtain it in an indirect way [7]). We will study techniques that support such bidirectional synchronization, namely the lenses approach proposed by Pierce *et al.* [8]. This will allow for less restrictive, while still guaranteeing safety, editions on the spreadsheet instances.

## References

[1] R. 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)*, July 2000.

[2] J. Cunha, J. Mendes, J. P. Fernandes, and J. Saraiva, "Embedding and evolution of spreadsheet models in spreadsheet systems," in *VLHCC '11: Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing*. Washington, DC, USA: IEEE Computer Society, 2011, to appear.

[3] G. Engels and M. Erwig, "ClassSheets: Automatic generation of spreadsheet applications from object-oriented specifications," in *ASE '05: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2005, pp. 124–133.

[4] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger, "Automatic generation and maintenance of correct spreadsheets," in *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*. New York, NY, USA: ACM, 2005, pp. 136–145.

[5] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert, "Visual specifications of correct spreadsheets," in *VLHCC '05: Procs. of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 189–196.

[6] L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva, "End-users productivity in model-based spreadsheets: An empirical study," in *IS-EUD '11: Proceedings of the Third International Symposium on End-User Development*, 2011, pp. 282–288.

[7] J. Cunha, M. Erwig, and J. Saraiva, "Automatically inferring ClassSheet models from spreadsheets," in *VLHCC '10: Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 93–100.

[8] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt, "Boomerang: resourceful lenses for string data," in *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York, NY, USA: ACM, 2008, pp. 407–419.