Towards an Evaluation of Bidirectional Model-Driven Spreadsheets

Jácome Cunha*, João Paulo Fernandes*†, Jorge Mendes*, João Saraiva*
*HASLab, INESC TEC & Universidade do Minho, Portugal
†Departamento de Engenharia Informática, Universidade do Porto, Portugal
{jacome,jpaulo,jorgemendes,jas}@di.uminho.pt

***Abstract*—Spreadsheets are widely recognized as popular programming systems with a huge number of spreadsheets being created every day. Also, spreadsheets are often used in the decision processes of profit-oriented companies. While this illustrates their practical importance, studies have shown that up to 90% of real-world spreadsheets contain errors.**

In order to improve the productivity of spreadsheet end-users, the software engineering community has proposed to employ model-driven approaches to spreadsheet development.

In this paper we describe the evaluation of a bidirectional model-driven spreadsheet environment. In this environment, models and data instances are kept in conformity, even after an update on any of these artifacts. We describe the issues of an empirical study we plan to conduct, based on our previous experience with end-user studies. Our goal is to assess if this model-driven spreadsheet development framework does in fact contribute to improve the productivity of spreadsheet users.

***Keywords*-Spreadsheets, Model-Driven Engineering, Software Evolution, Embedded DSLs, Model Inference, Bidirectional Transformations**

I. Introduction

Spreadsheets are widely recognized as extremely popular programming systems. Indeed, it is estimated that spreadsheets are used by hundreds of millions of people who create hundreds of millions of spreadsheets every year [1]. Also, most of these spreadsheets will be used in the future in the decision processes of profit-oriented companies [2]. While strategies for spreadsheet correctness may effectively save time and money, studies have shown that up to 90% of real-world spreadsheets contain errors [3].

In an effort to increase the productivity of using spreadsheets, the software-engineering community has acknowledged the need for model-driven spreadsheet development (MDSD). This approach highly resembles the way a civil engineer thinks of a house: first, a model is defined and thoroughly evolved, and only then a house is actually built.

For spreadsheets, the motivation of thinking at a higher abstraction degree lead to the development of spreadsheet modeling languages, one of the most widely accepted being the ClassSheet language [4]. Using ClassSheets, MDSD starts with the definition of a ClassSheet model, from which a concrete spreadsheet instance is derived. With respect to civil engineering, however, we want model and instance evolution to be more dynamic. Indeed, we will often want to change either the model or the corresponding instance, while desirably both are kept synchronized.

In order to achieve a practical model-driven spreadsheet environment, we have in the past proposed to embed ClassSheet models in spreadsheets themselves [5]. This feature was further fully integrated in a widely used spreadsheet system [6]. Our approach provided the first coherent and single environment for creating and evolving spreadsheet models while automatically obtaining conforming instances. Finally, we have recently studied in a rigorous way spreadsheet engineering in the reverse direction: we also allow for the data instances to evolve manually, while the corresponding models are automatically derived [7]. This is a feature that is currently being integrated in the tool presented in [6].

Adding the pieces of our work together, we have implemented in a spreadsheet system a bidirectional framework for MDSD. Although we believe that this model-driven approach can be beneficial in our context, this is a claim that needs to be verified in practice. Indeed, even having this approach proved successfully in other disciplines and in other software engineering fields, we want to provide empirical measurements of whether using models can increase the efficiency and effectiveness associated with using spreadsheets. This is precisely what we want to achieve by running a detailed study with real spreadsheet users.

We plan to capitalize the insights that we have taken from running previous usability studies with human users. Indeed, we have in the past assessed the productivity differences that arise from using different spreadsheet representations [8]: we have considered refactoring traditional spreadsheets and augmenting them with editing assistance features. Although we had then the same goals that we now chase, the fact is that all the different spreadsheet versions that we then tested were at the data instance level, whereas now it is the use of higher-level models that we believe can make a difference.

II. MDSheet: A Framework for Model-Driven Spreadsheet Engineering

In [5]–[7] we have introduced a framework for bidirectional evolution of spreadsheets. We have developed an

This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project refs. PTDC/EIA-CCO/108613/2008 and PTDC/EIA-CCO/120838/2010. The three first authors were funded by FCT grants SFRH/BPD/73358/2010, SFRH/BPD/46987/2008 and BI4-2011PTDC/EIA-CCO/108613/2008, respectively.

	A	B	C	D	E	F
1	**Budget**	**Year**	year=2010		...	
2	**Category**	Qnty	Cost	Total	...	Total
3	name=""	qnty=0	cost=0	total=qnty*cost	...	total=SUM(total)
4	...	...	...	...	...	...
5				total=SUM(total)	...	total=SUM(year.total)

(a) Budget spreadsheet model.

	A	B	C	D	E	F	G	H	I
1	**Budget**	**Year**	2010		**Year**	2011		...	
2	**Category**	Qnty	Cost	Total	Qnty	Cost	Total	...	Total
3	Travel	2	320	640	7	420	2940	...	3580
4	Accommodation	5	140	700	8	185	1480	...	2180
5	...	...	...	...	...	...	...	...	...
6				1340			4420	...	5760

(b) Budget spreadsheet instance.

Figure 1. Budget spreadsheet.

environment within a spreadsheet system allowing spreadsheet modelling and editing to be performed in the same framework. We have also implemented evolution steps on the model level that automatically co-evolve the data. The evolution of data and consequent co-evolution of the corresponding model was also developed. This guarantees that models and instances are always synchronized. In the next sections we explain this environment in more detail.

A. Spreadsheet Models

Erwig *et al.* [4] introduced the language of CLASSSHEETS to model spreadsheets at a high abstraction degree, thus allowing for spreadsheet reasoning to be performed at the conceptual level. CLASSSHEETS have a visual representation very similar to spreadsheets themselves: in Figure 1a, we present a possible model for a **Budget** spreadsheet, which we adapted from [4].[1]

This model holds three classes where data is to be inserted by end users: *i)* **Year**, with a default value of 2010, for the budget to accommodate multi-year information, *ii)* **Category**, for assigning a label to each expense and *iii)*, a relationship class where quantity and costs are registered. The actual spreadsheet may hold several repetitions of any of these elements, as indicated by the ellipsis. For each expense we record its quantity and its cost (with 0 as default value), and we calculate the total amount associated with it. Finally, (simple) summation formulas are used to calculate the global amount spent per year (cell D5), the amount spent per expense type in all years (cell F3) and the total amount spent in all years (cell F5) are also calculated.

Erwig *et al.* not only introduced CLASSSHEETS, but they also developed a tool - the Gencel tool [9] - that given a CLASSSHEET model generates an instance (*i.e.* a concrete spreadsheet) that conforms to the model. Figure 1b presents a possible spreadsheet as generated by Gencel given the CLASSSHEET shown in Figure 1a (and after an end user having manually introduced some data). In this particular case, the spreadsheet is used to record the annual budget for travel and accommodation expenses of an institution.

Since the spreadsheet is generated using all the information in the model, it is able of providing some correctness guarantees: formulas are kept consistent while new years are added, for example. Note also that, throughout the years, cost and quantity are registered for two types of expenses: travel and accommodation, and that formulas are used to calculate the total expense amounts.

B. Spreadsheet Evolution

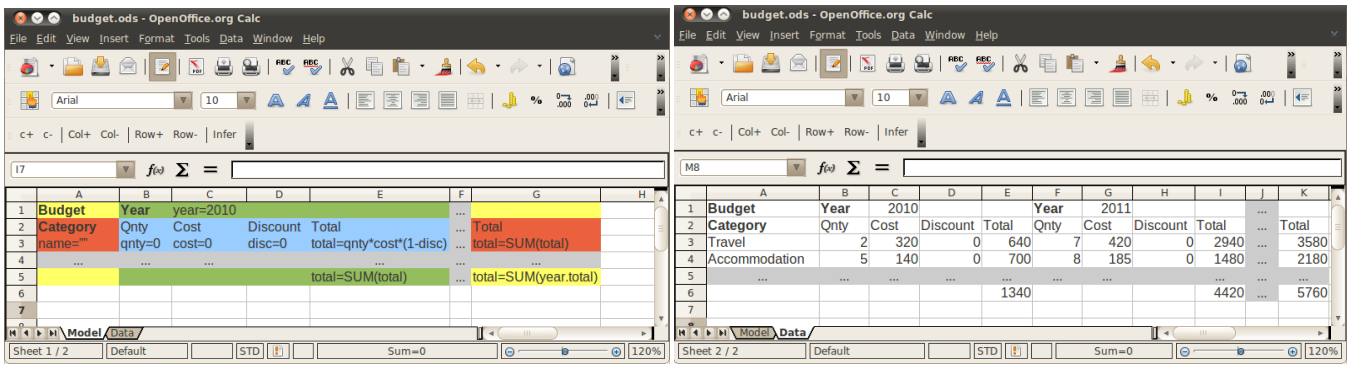
At the end of 2011, the spreadsheet of Figure 1b needs to be modified to accommodate 2012 data. Most spreadsheet users would typically take four steps to perform this task: *i)* insert three new columns; *ii)* copy all the labels ("Year", "Qnty", "Cost" and "Total"); *iii)* copy all the formulas (to compute the total amount spent per expense type in 2012, and the total expense for that same year) and *iv)* update all the necessary formulas in the last column to account for the new year information. More experienced spreadsheet users would possibly shortcut these steps by copy-inserting, for example, the 3-column block of 2011 and changing the label "2011" to "2012" in the copied block. Still, the range of the multi-year totals must be manually extended to include the new year information. In any of these situations, or in any combination of them, a conceptually unitary modification, *add year*, needs to be executed via an error-prone combination of steps.

This is precisely the main advantage of MDSD: it is possible to provide unitary transformations such as the addition of class instances (e.g., a year or a category) as one-step procedures, while all the structural impacts of such transformations are handled automatically (e.g., the involved formulas being automatically updated). This advantage is exploited to its maximum when the model and the instance are part of the same spreadsheet development environment, as it was proposed for OpenOffice in [5].[2] Besides automation, it is also guaranteed that this type of instance level operations does not affect the model-instance conformity binding.

There are, however, several situations in which the user prefers to change a spreadsheet instance (or a particular model) in such a way that, after the edit, it will no longer conforms to the previously defined model (or the respective instance). For example, if the user wants to add a column containing a possible expense discount for a particular year only, this is a trivial operation to perform at the data level which is actually not simple to perform at the model level. In [7] we have proposed a technique to automatically construct a new model based on the data evolution. In this case, the new model would keep the class for years with 3

[1] We assume colors are visible in the digital version of this paper.

[2] Actually, Figure 1 presents a CLASSSHEET model and a spreadsheet instance as defined in the embedding of CLASSSHEETS in spreadsheets [5].



(a) Model environment. (b) Data environment.

Figure 2. A bidirectional model-driven environment for the budget spreadsheet.

columns, while a new class for years with an extra discount column is introduced.

C. A Bidirectional Model-Driven Spreadsheet Environment

In this section we present a bidirectional framework that maintains the consistency between a model and its instance. By being bidirectional, it supports either manually evolving the spreadsheet instances, as we have described in the previous section, or editing the model instead. In any case, the correlated artefact is automatically co-evolved, so that their conformity relationship is always preserved.

In Figure 2, we present an overview of the bidirectional spreadsheet environment we propose. In Figure 2a, the embedded CLASSSHEET is presented in the Model worksheet while the data instance that conforms to it is given in Figure 2b, in the Data worksheet. Both worksheets contain buttons that perform the evolution steps at the model and instance levels.

Every time a (model or instance) spreadsheet evolution button is pressed, the system responds automatically. Indeed, it was built to propagate one update at a time and to be highly interactive by immediately giving feedback to users.

Also, our bidirectional spreadsheet engine makes some natural assumptions on the models it is able to manipulate and restricts the number of operations that are allowed on the instance side [7].

III. TOWARDS AN EVALUATION OF MDSHEET

A. Previous User Evaluation

Our research goal has long been to raise the productivity associated with the use of spreadsheets. Along these lines, we have proposed several techniques, of which two were selected to be assessed through user evaluation. Indeed, we have conducted an empirical study, described in [8], where we have compared the productivity, in terms of both efficiency and effectiveness, in accomplishing a series of tasks in three different spreadsheet representations.

The representations that were assessed included: *i) original* spreadsheets, i.e., spreadsheets in the traditional rectangular $m\ lines * n\ columns$ format; *ii) refactored* spreadsheets, where the original data is stored in tables obtained using the database normalization techniques that we proposed in [10]; *iii) visual* spreadsheets, that add edit assistance features to spreadsheets in order to guide users in introducing correct data [11].

We used a within-subjects design, where each participant received a task list for each of the three different spreadsheet representations. The tasks included entering data, modifying existing data and calculating new data based on the already existing spreadsheet information. Participants were encouraged to work as accurately and quickly as possible.

38 study participants started out by filling a background questionnaire so we could collect their area of study and previous experience with spreadsheets and other programming languages, and English comfort (Portuguese was their mother language). An introduction to the study was given orally in English, but this was explicitly not a tutorial for the different environments. Furthermore, the order of the spreadsheets received by each user was randomized, and the time limit for the study was set to two hours. A post-session questionnaire was handed which contained questions that could only be correctly answered by participants having understood the running representations.

From preparing and running our study, and from the analysis we produced out of it, we have drawn several insights, the most important one being that embedding database normalization techniques and visual editing assistance features in spreadsheets can bring benefits for spreadsheet users. Nevertheless, we could not find statistical evidence to support this claim.

Several study aspects, that we want to prevent in the study to conduct and that we would like to discuss at the workshop, may have contributed to the lack of statistical support: *a)* the fact that the study was performed in English; *b)* the fact that participants were college students studying

on majors other than engineering or computer science, with low spreadsheet knowledge, *c)* the fact that a tutorial was not given to participants.

B. *Evaluating* MDSHEET

With the user study that we propose ourselves to conduct, we now want to assess the potential productivity benefits of working under the MDSHEET environment. The main research question for which we seek an answer is: are users more effective and/or more efficient if they engage the bidirectional model-driven spreadsheet development of MDSHEET, or the approach still needs to be improved? If not, in which direction should we take it?

We plan to recruit a significant number of participants within our universities students, with the possibility for them to receive some gratification. Industry participants are also being considered. While we believe that the motivation for the study is clearly defined, several questions remain open and can benefit from detailed discussion:

- Should we introduce the CLASSSHEET language by running a pre-study tutorial?

We believe that not having conducted one such tutorial in our previous study had a significant negative impact on the conclusions we have drawn from it. Does this intuition propagate to our new context?

- Should we introduce the problem domain (a concrete pair model/instance) explicitly before the study?

Even resembling spreadsheets themselves, CLASSSHEETS are not straightforward to understand, especially for users outside the software engineering domain. Can this be alleviated by an explicit introduction?

- What should the participant profile be? Should we prefer software engineering/computer science students? Or instead choose students with other backgrounds?

A criterial selection of study participants is of crucial nature for our study, so it is very important for us to establish clearly the adequate profile.

- Is it reasonable to impose a time limit on the study? Even more interesting, should the same task list be given to different user groups, each of which with a different time limit?

We believe that given the sufficient amount of time, most users would eventually end up by succeeding in all the requested tasks. We also know, however, that real world professional life imposes limited time for a concrete assignment. Should we try to establish a threshold for users effectiveness, given different time amounts?

- How many spreadsheets should we use? Should we run a within-subject study?

We believe that these are all important questions that can greatly benefit from the type of discussions that are scheduled for the workshop.

IV. CONCLUSION

We propose to conduct an empirical study with real users to evaluate the potential productivity benefits of using the MDSHEET framework. While most of the technical contributions of the framework have already been validated, we believe that it still lacks an evaluation phase with spreadsheet end-users. Regarding the study to conduct, several issues remain open and may benefit from being discussed as widely as possible.

REFERENCES

[1] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," *VLHCC'05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 207–214, 2005.

[2] W. L. Winston, "Executive education opportunities," *OR/MS Today*, vol. 28(4), pp. 8–10, 2001.

[3] K. Rajalingham, D. Chadwick, and B. Knight, "Classification of spreadsheet errors," *European Spreadsheet Risks Interest Group (EuSpRIG)*, 2001.

[4] 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.

[5] J. Cunha, J. Mendes, J. P. Fernandes, and J. Saraiva, "Embedding and evolution of spreadsheet models in spreadsheet systems," in *VL/HCC'11: IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE CS, 2011, pp. 179–186.

[6] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "MDSheet: A framework for model-driven spreadsheet engineering," in *ICSE'12: Proceedings of the 34rd International Conference on Software Engineering*. ACM, 2012, pp. 1412–1415.

[7] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva, "Bidirectional transformation of model-driven spreadsheets," in *ICMT '12: 5th International Conference on Model Transformation*. Springer LNCS, 2012, (to appear).

[8] 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*. LNCS, 2011, pp. 282–288.

[9] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein, "Gencel: a program generator for correct spreadsheets," *J. Funct. Program*, vol. 16, no. 3, pp. 293–325, 2006.

[10] J. Cunha, J. Saraiva, and J. Visser, "From spreadsheets to relational databases and back," in *PEPM'09: Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. ACM, 2009, pp. 179–188.

[11] ——, "Discovery-based edit assistance for spreadsheets," in *VL/HCC'09: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE CS, 2009, pp. 233–237.