Extension and Implementation of ClassSheet Models

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

***Abstract*—In this paper we explore the use of models in the context of spreadsheet engineering. We review a successful spreadsheet modeling language, whose semantics we further extend. With this extension we bring spreadsheet models closer to the business models of spreadsheets themselves.**

An addon for a widely used spreadsheet system, providing bidirectional model-driven spreadsheet development, was also improved to include the proposed model extension.

***Index Terms*—Spreadsheets, Model-Driven Engineering, Bidirectional Software Evolution, Embedded Domain-Specific Languages**

I. INTRODUCTION

Spreadsheets play an important role in software organizations. Indeed, in large software organizations, spreadsheets are not only used to define sheets containing data and formulas, but also to collect information from different systems, to adapt data coming from one system to the format required by another, to perform operations to enrich or simplify data, etc. In fact, over time many spreadsheets turn out to be used for storing and processing increasing amounts of data and supporting increasing numbers of users. Unfortunately, spreadsheet systems provide poor support for modularity, abstraction, and transformation, thus, making the maintenance, update and evolution of spreadsheets a very complex and error-prone task.

In this paper, we consider the use of models for guiding users in introducing correct data. Our technique is developed as part of a global environment for the evolution of spreadsheets. We use spreadsheet models, namely *ClassSheet* models [1], to express the business logic of the underlying spreadsheet data. These models are used to generate (spreadsheet) instances that guide end users to introduce correct data: the generated spreadsheet instances have implanted the underlying *ClassSheet* model. This may be considered an advanced spreadsheet mechanism, which guarantees that an user update always produces a spreadsheet instance that conforms to the defined model. Because *ClassSheet* models have a visual representation that is very similar to spreadsheets themselves, we embed *ClassSheets* in spreadsheet systems. This embedding has an important property: it provides a model-driven environment for spreadsheets where users can not only edit the spreadsheet data instance but also evolve the *ClassSheet* model. We have presented the embedding and evolution of *ClassSheet* models in [2], [3] and a model-driven spreadsheet environment in [4].

This paper presents our ongoing work on model-driven spreadsheet environments. Its goal is two-fold:

- First, we extend the original definition of *ClassSheet* models in order to have a more powerful formalism to model the business logic of spreadsheets. We extend *ClassSheets* with *type constraints* and *regular expressions*, and we review the use of *primary and foreign keys* of relational models.
- Second, we present the details of the implementation of these extensions in the *ClassSheet* embedding [2]. This embedding is part of an existing addon for bidirectional model-driven spreadsheet development that permits to model spreadsheets and co-evolve both models and data.

This paper is organized as follows: Section II gives a brief introduction to *ClassSheet* models and presents the proposed extensions. In Section III we describe how the proposed extension were integrated under a global framework for model-driven spreadsheet development. Finally, Section IV concludes the paper.

II. CLASSSHEET MODELS AND EXTENSIONS

ClassSheets [1] 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). Thus, we get an abstraction layer on top of spreadsheets which allows to understand more easily the business logic and to think more clearly about spreadsheet transformation. Also, dividing models into several classes, we add some modularity to spreadsheet development.

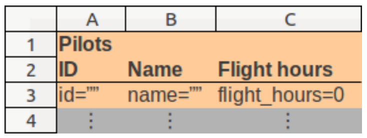
Using *ClassSheet* models, it is possible to define spreadsheet tables and to give them names, to define labels for the table's columns, to specify the types of the values such columns may contain and also the way the table expands (*e.g.*, horizontally or vertically).

Besides a textual (and formal) definition, *ClassSheets* also have a visual representation which very much resembles spreadsheets themselves [5]. In [2], we have embedded such

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 also supported by FCT grants SFRH/BPD/73358/2010, SFRH/BPD/46987/2008 and BI4-2011PTDC/EIA-CCO/108613/2008, respectively.

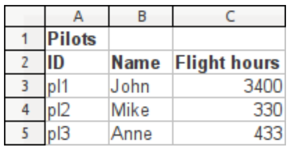
visual model representation that mimics the well-known embedding of a domain specific language in a general purpose one. Next, we briefly describe *ClassSheet* models through this embedding/notation. After that, we introduce several extensions to the *ClassSheet* models and we present the details of its embedding in a widely used spreadsheet system.

In order to illustrate *ClassSheets* we present in Figure 1 an embedded/visual ClassSheet model (Figure 1a) and one of its possible instances (Figure 1b).



	A	B	C
1	Pilots		
2	ID	Name	Flight hours
3	id=""	name=""	flight_hours=0
4	⋮	⋮	⋮

(a) Pilot *ClassSheet* model.



	A	B	C
1	Pilots		
2	ID	Name	Flight hours
3	pl1	John	3400
4	pl2	Mike	330
5	pl3	Anne	433

(b) Pilot table.

Fig. 1: Registering pilot information.

Considering the spreadsheet development environment of [4], it is possible to automatically obtain a data instance (that is empty) once a model is defined, and also to infer a model for a particular data spreadsheet. Furthermore, it is possible to manually evolve either a model or an instance of it such that the correlated artifact is automatically updated.

The *ClassSheet* model in Figure 1 represents pilots, where a pilot is defined by an **ID**, **Name** and **Flight hours**. In row 3, it is defined the type, and the default value, associated with each column: columns A and B hold strings (with an empty string as default value), and column C holds integer values (with 0 as default). The fourth row of the model contains vertical ellipses in all columns. This means that it is possible for these columns to expand vertically: the tables that conform to this model can have as many rows (entries) as needed.

Reusing the *ClassSheet* table we built before, we can now model a table to register concrete flights by an airline company, as shown in the bottom-right part of Figure 2.

The colors[1] in the model are used to distinguish the different entities represented, namely, *pilots*, *planes*, *references to pilots* in the scheduling table, *reference to planes* in the scheduling table and the *flight scheduling* itself.

A. Extending ClassSheet Models

ClassSheet models are a powerful formalism to specify the business logic of spreadsheet data. However, they are very syntactic/visual oriented in the sense that they mainly specify layout rules, simple formulas relating components of the models and repetition blocks of columns/rows within a spreadsheet. The semantic guarantees provided by *ClassSheet* models are very poor: they provide a default value for cells that can be seen as a very trivial type system.

In this section we present three semantic extensions to *ClassSheet* models, which make them closer to the business models of spreadsheets.

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

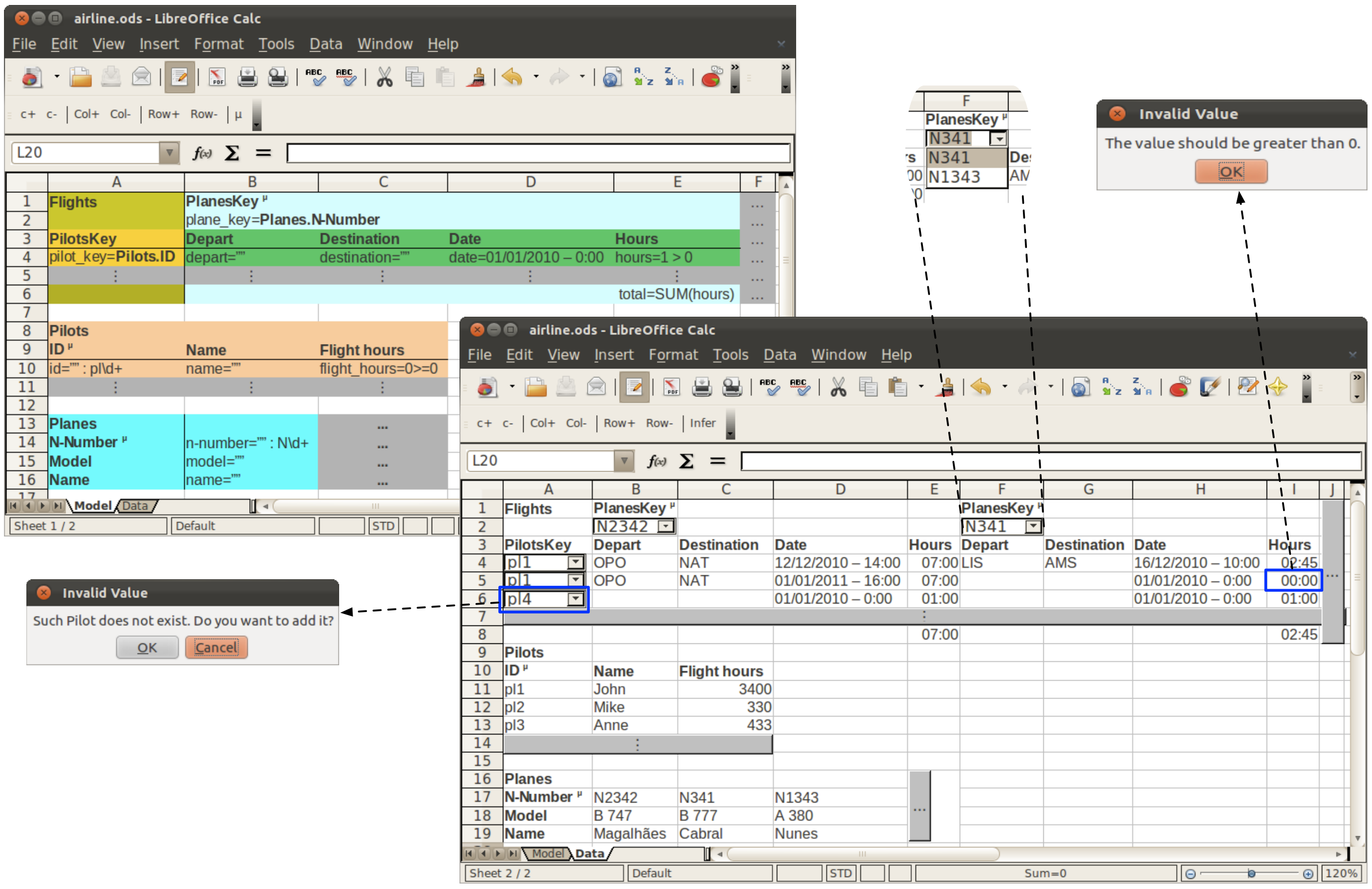
- *Type Constraints* — Like spreadsheet systems, *ClassSheets* do not have a strong type system. *ClassSheets* define the default value of cells, but not their type, nor any value constraint. We extend *ClassSheets* with a very simple type constraint notation. With this extension, we can specify that the flight hours cells must contain positive values, for example.
- *Regular Expressions* — It is common in spreadsheets that some cells contain string values that follow some pattern. For example, the pilot id cells have to start with the pattern "pl" followed by a natural number. To force cell values to follow a predefined pattern we extend *ClassSheets* with regular expressions.
- *Relational Model* — Usually spreadsheet tables include information where one column is a key that defines the values of other columns in the same table. Figure 1b shows such an example, where the pilot id determines the name and flight hours. Moreover, the pilot id can be referenced in another table, for example, defining the company flights (this is presented in Figure 2). To prevent data inconsistencies we extend *ClassSheets* with the notion of primary key and foreign key from databases. The theoretical details of this extension are presented in [6], and in this paper we focus on its embedding as presented in the next section.

In Figure 3 we present the formal definition of *ClassSheets* and the proposed extensions (in red).

$$
\begin{array}{lcl}
\prec \in Lesser & ::= & < \mid \leqslant \\
\succ \in Greater & ::= & > \mid \geqslant \\
t \in Val & ::= & = \varphi \mid \succ \varphi \mid \prec \varphi \mid \succ \varphi, \prec \varphi \mid \,!\, t \\
e \in RegExp & ::= & \varepsilon \mid \varphi \mid \backslash d \mid \backslash w \mid e* \mid e+ \mid e \shortparallel e \mid ee \\
f \in Fml & ::= & \varphi \mid \varphi : e \mid \varphi \ t \mid n.a \mid \varphi(f, \ldots, f) \\
b \in Block & ::= & \varphi \mid \varphi^{\mu} \mid a = f \mid b \shortmid b \mid b \,\hat{}\, b \\
l \in Lab & ::= & h \mid v \mid .n \\
h \in Hor & ::= & \underline{n} \mid \underline{\shortmid n} \\
v \in Ver & ::= & \shortmid n \mid \underline{\shortmid n} \\
c \in Class & ::= & l : b \mid l : b^{\downarrow} \mid c \,\hat{}\, c \\
s \in Sheet & ::= & c \mid c^{\rightarrow} \mid s \shortmid s
\end{array}
$$

Fig. 3: The *ClassSheet* language.

The first extension we introduce is the type constraints. The type of a cell is infered from its default value. If the default value is a string then the type of that cell is string. The same happens for whole and decimal numbers, date, and time values. In fact, these are the types allowed in spreadsheets. Based on the infered type, it is possible to apply restrictions on the values of that type: $\varphi\ t$, where $t ::= = \varphi \mid \succ \varphi \mid \prec \varphi \mid \succ \varphi \prec \varphi \mid \,!\, t$. These restrictions allow users to specify values greater (or lesser) than a certain limit, *e.g.*, $\geqslant 0$, values between a lower and an upper limit, *e.g.*, $\geqslant 10, < 20$, and negation of ranges, *e.g.*, $! > 20$. Also, they are already available in common spreadsheet systems, but they are not very widespread among basic users and require


airline.ods - LibreOffice Calc

	A	B	C	D	E	F
1	Flights	PlanesKey µ				…
2		plane_key=Planes.N-Number				…
3	PilotsKey	Depart	Destination	Date	Hours	…
4	pilot_key=Pilots.ID	depart=""	destination=""	date=01/01/2010 – 0:00	hours=1 > 0	…
5	⋮	⋮	⋮	⋮	⋮	…
6					total=SUM(hours)	…
7						
8	Pilots					
9	ID µ	Name	Flight hours			
10	id="" : pl\d+	name=""	flight_hours=0>=0			
11	⋮	⋮	⋮			
12						
13	Planes		…			
14	N-Number µ	n-number="" : N\d+	…			
15	Model	model=""	…			
16	Name	name=""	…			

Invalid Value

Such Pilot does not exist. Do you want to add it?

OK Cancel

airline.ods - LibreOffice Calc

	A	B	C	D	E	F	G	H	I
1	Flights	PlanesKey µ				PlanesKey µ			
2		N2342				N341			
3	PilotsKey	Depart	Destination	Date	Hours	Depart	Destination	Date	Hours
4	pl1	OPO	NAT	12/12/2010 – 14:00	07:00	LIS	AMS	16/12/2010 – 10:00	02:45
5	pl1	OPO	NAT	01/01/2011 – 16:00	07:00			01/01/2010 – 0:00	00:00
6	pl4			01/01/2010 – 0:00	01:00			01/01/2010 – 0:00	01:00
7					⋮				
8					07:00				02:45
9	Pilots								
10	ID µ	Name	Flight hours						
11	pl1	John	3400						
12	pl2	Mike	330						
13	pl3	Anne	433						
14	⋮								
15									
16	Planes								
17	N-Number µ	N2342	N341	N1343	…				
18	Model	B 747	B 777	A 380					
19	Name	Magalhães	Cabral	Nunes					

F
PlanesKey µ
N341
N341
N1343

Invalid Value

The value should be greater than 0.

OK

File Edit View Insert Format Tools Data Window Help
c+ c- Col+ Col- Row+ Row- µ
c+ c- Col+ Col- Row+ Row- Infer
L20
Model Data
Sheet 1 / 2
Sheet 2 / 2
Default
STD
Sum=0
120%

Fig. 2: A model-driven spreadsheet to manage an airline company.

some effort from the user to define them. Introducing these restrictions directly on the model, it is easier to define them and to apply them to a set of cells.

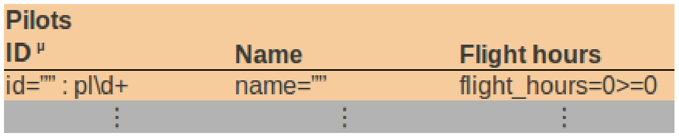
The second extension we introduce is the use of regular expressions to define the allowed values of a cell: $\varphi : e$, where $e ::= \varepsilon \mid \varphi \mid \backslash d \mid \backslash w \mid e* \mid e+ \mid e \shortparallel e \mid ee$ (the common regular expressions language [7]). They are useful for more experienced users and allow a more fine-tuned specification of the allowed values.

Although the relational extensions (φ^{μ} and $n.a$) were already presented in [6], we highlight them here since we explain in this paper their implementation in the embedding.

An example containing these extensions is present in Figure 4. It shows the spreadsheet-like notation for the pilots *ClassSheet* model, where the pilot id is a primary key of that table and its value must match the regular expression `pl\d+`. The pilot name must be a string since the default value is an empty string, and the flight hours must contain positive numbers only.

III. Implementation of Extended Spreadsheet Models

In our previous work, we have proposed the embedding of *ClassSheet* models in spreadsheet systems [2], allowing simple



Pilots		
ID µ	Name	Flight hours
id="" : pl\d+	name=""	flight_hours=0>=0
⋮	⋮	⋮

Fig. 4: An extended *ClassSheet* model specifying the pilots table.

evolution of models so that and the structure of their data instances is automatically updated. That work was improved to support bidirectional co-evolution [8], so that changes in data instances would also automatically reflect on the corresponding models. Furthermore, these techniques where included as part of the widely used spreadsheet system OpenOffice [4]. This last feature facilitates the development of the spreadsheet business logic enabling the user to do transformations on the data instance that would otherwise be too complex to perform when evolving just the model. Also, it may provide a better perspective for certain transformations.

The work that we now propose improves the system of [4] since it includes the *ClassSheet* extension described in this paper. In the remaining of this section we introduce in greater detailed how each extension is implemented in such system.

A. Type Constraints

As the type constraints used are the ones available in spreadsheet systems, the *cell validity rules* provided by those systems were used to support this extension. Moreover, we lift these rules to the model level, providing an easier way to define them (using the language presented in the previous section). With this feature, when a user introduces a wrong value, the spreadsheet system provides a notification so the user can take the necessary steps to fix the error (*i.e.*, introduce a correct value). This notification is illustrated in Figure 2 (top-right) where a value not in conformity with the model was inserted in cell `I5`.

Type constraints are straightforwardly embedded in the data instances using the cell validity mechanism and it is the spreadsheet system itself that checks the value and raises the error message. We believe it is preferable to use the spreadsheet system mechanisms as much as possible since the user is already familiar with them.

B. Regular Expressions

Since common spreadsheet systems do not provide validity rules for regular expression, an interpreter is included with the addon. The value of the cell is interpreted every time it is changed, and a notification is provided if the value does not conform to the regular expression, as if it were a type constraint. This extension is similar with the previous one, but the with its own acceptance function. Moreover, the notification system tries to mimic the default cell validity feature providing the same familiar environment to the user.

C. Relational Model

Primary keys are identified adding an annotation to the cell, using $^{\mu}$ (as defined by the extended *ClassSheet* reference in Figure 3). When this annotation is present, the cells in that column or row are interpreted as a primary key (or unique value) and the system warns the user when introducing a repeated value. The verification of the value is done within the system and not with the help of formulas, so that the user does not change the formula and introduces wrong data. When a repeated value is detected, an error message is shown to the user, so that value can be changed.

Foreign keys are defined in the model using references in formulas. On the data sheet, those cells contain combo boxes where users can select a valid value from the ones that are referenced, or add a new value (Figure 2, top-right, cell `F2`). When the latter action is performed, a message is shown to the user indicating that the key is not present and offers an option to create a new entry with that key (Figure 2, bottom-left, cell `A6` and associated message). If the user prefers to not add that entry, it is possible to cancel the action and select an existing key from the list. This introduction method is an improvement on the system presented in [6], since it uses more components from the spreadsheet system, like the combo boxes and the ability to insert new entries just by clicking on the notification, providing a more user-friendly environment.

IV. Conclusions

In this work we extended *ClassSheets* with several semantic mechanisms. The already existing naive relational extension is now a powerful tool that guides users in introducing correct data, that avoids the corruption of the spreadsheet data. Moreover, the type constraints we introduce allow the user to lift the cell oriented validity rules to a more proficient level since they have been directly integrated in the model.

Nevertheless, the data integration is still done using commonly available spreadsheet mechanisms so the user gets as little differences as possible from the system he/she is used to. Also, using regular expressions, a widely used specification language, the user can express in an exact way the values that are allowed for particular cells. The full integration of these features in the existing embedding provides users with more power than ever to adjust models to the required business logic.

Finally, details about the integration of the model extension within the existing framework were provided, indicating how the user can interact with the new features.

We believe that the general framework we obtain in the end is in a mature stage, and thus we would like to evaluate it with real spreadsheet users. Indeed, we would like to have empirical evidences that this framework indeed improves spreadsheet users' productivity. For this, we are preparing a usability study, which we are trying to improve with feedback from the human-interaction community [9].

The work described in this paper was conducted under the *Spreadsheets as a Programming Paradigm* research project:

`http://ssaapp.di.uminho.pt`

References

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

[2] J. Cunha, J. Mendes, J. P. Fernandes, and J. Saraiva, "Embedding and evolution of spreadsheet models in spreadsheet systems," in *VL/HCC'11: IEEE Symp. on Visual Languages and Human-Centric Computing*. IEEE Computer Society, 2011, pp. 186–201.

[3] J. Cunha, J. Visser, T. Alves, and J. Saraiva, "Type-safe evolution of spreadsheets," in *FASE'11/ETAPS'11: Proc. of the 14th International Conference on Fundamental Approaches to Software Engineering*. Springer-Verlag, 2011, pp. 186–201.

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

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

[6] J. Cunha, J. P. Fernandes, and J. Saraiva, "From Relational ClassSheets to UML+OCL," in *SAC'12: the Software Engineering Track at the 27th Annual ACM Symposium On Applied Computing*. ACM, 2012, (to appear).

[7] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[8] 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*, 2012, (to appear).

[9] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva, "Towards an Evaluation of Bidirectional Model-driven Spreadsheets," in *USER'12: User evaluation for Software Engineering Researchers*, 2012, (to appear).