Model-based Programming Environments for Spreadsheets

Jácome Cunha[a,c], Jorge Mendes[a,c], João Saraiva[a], Joost Visser[b]

[a] *HASLab / INESC TEC, Universidade do Minho, Portugal*
[b] *Software Improvement Group & Radboud University Nijmegen, The Netherlands*
[c] *CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal*

Abstract

Spreadsheets can be seen as a flexible programming environment. However, they lack some of the concepts of regular programming languages, such as structured data types. This can lead the user to edit the spreadsheet in a wrong way and perhaps cause corrupt or redundant data.

We devised a method for extraction of a relational model from a spreadsheet and the subsequent embedding of the model back into the spreadsheet to create a model-based spreadsheet programming environment. The extraction algorithm is specific for spreadsheets since it considers particularities such as layout and column arrangement. The extracted model is used to generate formulas and visual elements that are then embedded in the spreadsheet helping the user to edit data in a correct way.

We present preliminary experimental results from applying our approach to a sample of spreadsheets from the EUSES Spreadsheet Corpus.

Finally, we conduct the first systematic empirical study to assess the effectiveness and efficiency of this approach. A set of spreadsheet end users worked with two different model-based spreadsheets, and we present and analyze here the results achieved.

Keywords: Spreadsheets, Model-Driven Engineering, Model-Driven Spreadsheets, Empirical Validation

1. Introduction

Developments in programming languages are changing the way in which we construct programs: naive text editors are now replaced by powerful programming language environments which are specialized for the programming language under consideration and which help the user throughout the editing process. Helpful features like highlighting keywords of the language or maintaining a beautified indentation of the program being edited are now provided by several text editors. Recent advances in programing languages extend such naive editors to powerful language-based environments [1, 2, 3, 4, 5, 6]. Language-based environments use *knowledge* of the programming language to provide the users with more powerful mechanisms to develop their programs. This knowledge is based on the *structure* and the *meaning* of the language. To be more precise, it is based on the syntactic and (static) semantic characteristics of the language. Having this knowledge about a language, the language-based environment is not only able to highlight keywords and beautify programs, but it can also detect features of the programs being edited that, for example, violate the properties of the underlying language. Furthermore, a language-based environment may also give information to the user about properties of the program under consideration. Consequently, language-based environments guide the user in writing correct and more reliable programs.

Spreadsheet systems can be viewed as programming environments for non-professional programmers, the so-called *end-user* programmers . In order to improve end-user productivity, modern spreadsheet systems offer some of the mechanisms found in programming environments, as, for instance, auto completion of names/cells, or warnings when mismatching data types. These mechanisms, however, are very limited: they focus on the *structure* of the data,

Email addresses: jacome@di.uminho.pt (Jácome Cunha), jorgemendes@di.uminho.pt (Jorge Mendes), jas@di.uminho.pt (João Saraiva), j.visser@sig.eu (Joost Visser) and not on the *meaning* of the spreadsheet data. As a consequence, the end-user guidance provided by spreadsheets systems is very weak when compared to modern programming environments.

This paper extends previous work on generating powerful spreadsheet environment [7, 8]: we propose a technique to enhance a spreadsheet system with mechanisms to guide end users to introduce correct data. A background process adds formulas and visual objects to an existing spreadsheet, based on a relational database schema. To obtain this schema, or model, we follow the approach used in language-based environments: we use the *knowledge* about the data already existing in the spreadsheet to guide end users in introducing correct data. The knowledge about the spreadsheet under consideration is based on the *meaning* of its data that we infer using data mining and database normalization techniques.

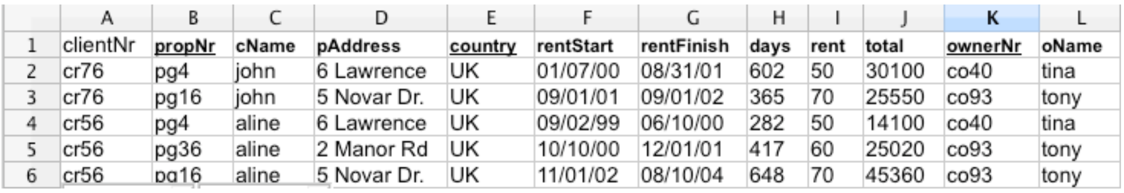
Data mining techniques specific to spreadsheets are used to infer *functional dependencies* from the spreadsheet data. These functional dependencies define how certain spreadsheet columns determine the values of other columns. Database normalization techniques, namely the use of normal forms [9], are used to eliminate redundant functional dependencies, and to define a relational database model. Knowing the relational database model induced by the spreadsheet data, we construct a new spreadsheet environment that not only contains the data of the original one, but that also includes advanced features which provide information to the end user about correct data that can be introduced. We consider several types of advanced features: *auto-completion of column values*, *non-editable columns*, *safe deletion of rows*, *key columns* and *reference columns*.

Besides presenting in this paper an extension of our previous work on generating spreadsheet programming environments, we also include in this paper both the first experimental results validating the proposed techniques, which are obtained by considering a large set of spreadsheets included in the EUSES Spreadsheet Corpus [10], and an empirical study evaluating the effectiveness and efficiency of the model-based spreadsheets. Both experiments show that our techniques work not only for database-like spreadsheets, like the example we will use throughout the paper, but they work also for realistic spreadsheets defined in other contexts (for example, inventory, grades or modeling).

This paper is organized as follows. Section 2 presents an example used throughout the paper. Section 3 presents our algorithm to infer functional dependencies and how to construct a relational model. Section 4 discusses how to embed assisted editing features into spreadsheets. A preliminary evaluation of our techniques is present in Section 5. In Section 6 we present an empirical validation of the spreadsheets we generate. Section 7 discusses related work and Section 8 concludes the paper.

2. Spreadsheet Programming Environments

In order to present our approach we shall consider the following example taken from [11] and modeled in a spreadsheet as shown in Figure 1.



	A	B	C	D	E	F	G	H	I	J	K	L
1	clientNr	propNr	cName	pAddress	country	rentStart	rentFinish	days	rent	total	ownerNr	oName
2	cr76	pg4	john	6 Lawrence	UK	01/07/00	08/31/01	602	50	30100	co40	tina
3	cr76	pg16	john	5 Novar Dr.	UK	09/01/01	09/01/02	365	70	25550	co93	tony
4	cr56	pg4	aline	6 Lawrence	UK	09/02/99	06/10/00	282	50	14100	co40	tina
5	cr56	pg36	aline	2 Manor Rd	UK	10/10/00	12/01/01	417	60	25020	co93	tony
6	cr56	pg16	aline	5 Novar Dr.	UK	11/01/02	08/10/04	648	70	45360	co93	tony

Figure 1: A spreadsheet representing a property rental system.

This spreadsheet contains information related to a housing rental system. It gathers information about clients, owners, properties, prices and rental periods. The name of each column gives a clear idea of the information it represents. We extend this example with three additional columns, named $days$ (that computes the total number of rental days by subtracting the column $rentStart$ to $rentFinish$), $total$ (that multiplies the number of rental days by the rent per day value, $rent$) and $country$ (that represents the property's country). As usually in spreadsheets, the columns $days$ and $rent$ are expressed by formulas.

This spreadsheet defines a valid model to represent the information of the rental system. However, it contains redundant information: the displayed data specifies the house rental of two clients (and owners) only, but their names are included five times, for example. This kind of redundancy makes the maintenance and update of the spreadsheet complex and error-prone. A mistake is easily made, for example, by mistyping a name, thus corrupting the data on the spreadsheet.

Two common problems occur as a consequence of redundant data: *update anomalies* and *deletion anomalies* [12]. The former problem occurs when we change information in one place but leave the same information unchanged in the other places. The problem also occurs if the update is not performed exactly in the same way. In our example, this happens if we change the rent of property number `pg4` from `50` to `60` only in one row and leave the others unchanged, for example. The latter problem occurs when we delete some data and lose other information as a side effect. For example, if we delete row 5 in the our example all the information concerning property `pg36` is lost.

The database community has developed techniques, such as data normalization, to eliminate such redundancy and improve data integrity [12, 13]. Database normalization is based on the detection and exploitation of functional dependencies inherent in the data [14]. Can we leverage these database techniques for spreadsheets systems so that the system eliminates the update and deletion anomalies by guiding the end user to introduce correct data? Based on the data contained in our example spreadsheet, we would like to discover the following functional dependencies which represent the five entities involved in our house rental system: *countries*, *clients*, *owners*, *properties*, and *rents* themselves:

$$
\begin{array}{l}
\mathit{country} \ \rightharpoonup \emptyset \\
\mathit{clientNr} \rightharpoonup \mathit{cName} \\
\mathit{ownerNr} \rightharpoonup \mathit{oName} \\
\mathit{propNr} \ \ \rightharpoonup \mathit{pAddress}, \mathit{rent}, \mathit{ownerNr} \\
\mathit{country}, \mathit{clientNr}, \mathit{propNr}, \mathit{rentStart}, \mathit{rentFinish} \rightharpoonup \mathit{days}, \mathit{total}
\end{array}
$$

A functional dependency $A \rightharpoonup B$ means that if we have two equal inhabitants of A, then the corresponding inhabitants of B are also equal. For instance, the client number functionally determines his/her name, since no two clients have the same number. The right hand side of a functional dependency can be an empty set. This occurs, for example, in the *country* functional dependency. Although these are extreme cases for functional dependencies, they are formally valid [15]. To better explain their existence, let us consider a table in a database with a single attribute *country*. To follow the guidelines of well-designed databases, this attribute would probably be primary key. The only functional dependency that holds in such a table is $\mathit{country} \rightharpoonup \emptyset$. This means that all the values in this table must be unique. In fact, as we will see in a few paragraphs, these cases will result in relational database tables where all the attributes are key, that is, all the entries in the table are unique.

Using these functional dependencies it is possible to construct a relational database schema. Each functional dependency is translated into a table where the attributes are the ones participating in the functional dependency and the primary key is the left hand side of the functional dependency. In some cases, foreign keys can be inferred from the schema. A foreign key (FK) is a set of attributes within one relation that matches the primary key of some relation. The relational database schema can be normalized in order to eliminate data redundancy. A possible normalized relational database schema created for the house rental spreadsheet is presented below.

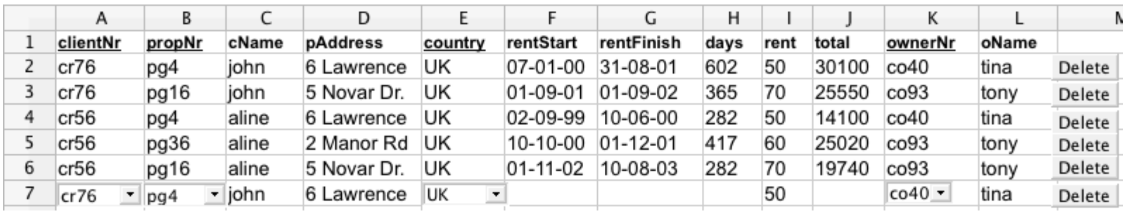
$$
\begin{array}{l}
\underline{\mathit{country}} \\
\underline{\mathit{clientNr}}, \mathit{cName} \\
\underline{\mathit{ownerNr}}, \mathit{oName} \\
\underline{\mathit{propNr}}, \mathit{pAddress}, \mathit{rent}, \#\mathit{ownerNr} \\
\underline{\#\mathit{country}, \#\mathit{clientNr}, \#\mathit{propNr}, \mathit{rentStart}, \mathit{rentFinish}}, \mathit{days}, \mathit{total}
\end{array}
$$

This database schema defines a table for each of the entities described before. Note that the attributes marked with the symbol # are *foreign keys* to other tables and underlined attributes represent the key of the table. We explore this in Section 3. As we said, the first and the last tables are formed only by key attributes because the dependencies used to create them had no right-hand side.

In the next two sub-sections we will use all this information to create two different spreadsheets. The first one, termed *schema-aware*, is intended to keep the same structure as the original, but still with some features to guide users inputting correct data. The second one, named *refactored*, will have a different structure from the original one, since repeated information is removed and “entities” are factored out.

2.1. Schema-aware Spreadsheet Environment

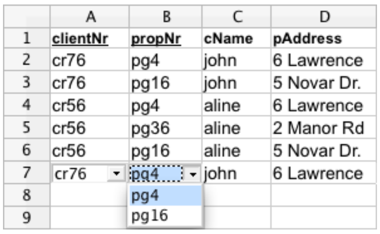
Having defined a relational database schema we would like to construct a spreadsheet environment that respects that relational model, as shown in Figure 2, without changing the initial layout of the spreadsheet. This spreadsheet is from now on termed *schema-aware*.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	clientNr	propNr	cName	pAddress	country	rentStart	rentFinish	days	rent	total	ownerNr	oName	
2	cr76	pg4	john	6 Lawrence	UK	07-01-00	31-08-01	602	50	30100	co40	tina	Delete
3	cr76	pg16	john	5 Novar Dr.	UK	01-09-01	01-09-02	365	70	25550	co93	tony	Delete
4	cr56	pg4	aline	6 Lawrence	UK	02-09-99	10-06-00	282	50	14100	co40	tina	Delete
5	cr56	pg36	aline	2 Manor Rd	UK	10-10-00	01-12-01	417	60	25020	co93	tony	Delete
6	cr56	pg16	aline	5 Novar Dr.	UK	01-11-02	10-08-03	282	70	19740	co93	tony	Delete
7	cr76	pg4	john	6 Lawrence	UK				50		co40	tina	Delete

Figure 2: A spreadsheet with auto-completion based on relational tables.

For example, this spreadsheet would not allow the user to introduce two different properties with the same property number *propNr*. Instead, we would like that the spreadsheet offers to the user a list of possible properties, such that he/she can choose the value to fill in the cell. Figure 3 shows a possible spreadsheet environment where possible properties can be chosen from a *combo box*.



	A	B	C	D
1	clientNr	propNr	cName	pAddress
2	cr76	pg4	john	6 Lawrence
3	cr76	pg16	john	5 Novar Dr.
4	cr56	pg4	aline	6 Lawrence
5	cr56	pg36	aline	2 Manor Rd
6	cr56	pg16	aline	5 Novar Dr.
7	cr76	pg4	john	6 Lawrence
8		pg4		
9		pg16		

Figure 3: Selecting possible values of columns using a combo box.

Using the relational database schema we would like to have a spreadsheet system offering the following features:

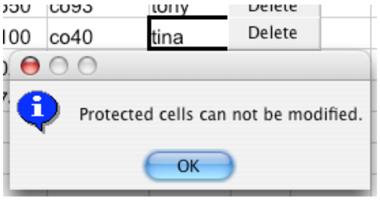
Auto-completion of Column Values. The columns corresponding to primary keys in the relational model determine the values of other columns; we want the spreadsheet environment to be able to automatically fill those columns provided the end user defines the value of the primary key.

For example, the value of the property number (*propNr*, column `B`) determines the values of the address (*pAddress*, column `D`), rent per day (*rent*, column `I`), and owner number (*ownerNr*, column `K`). Consequently, the spreadsheet environment should be able to automatically fill in the values of the columns `D`, `I` and `K`, given the value of column `B`. Since *ownerNr* (column `K`) is a primary key of another table, transitively the value of *oName* (column `L`) is also defined. This auto-completion mechanism has been implemented and is presented in the spreadsheet environment of Figure 2.

Non-Editable Columns. Columns that are part of a table (as the ones previously computed from the functional dependencies), but not part of its primary key must not be edited without some control. For example, column `L` is part of the owner table but it is not part of its primary key. Thus, it must be protected from being edited arbitrarily. As we saw in the previous feature, they are edit through auto-completion, being this the only way to change their values.

The primary key of a table must not be randomly edited either since it can destroy the dependency. This feature prevents the end user from introducing potentially incorrect data and, thus, producing update anomalies. Still, such columns can be edited using the available combo boxes, as introduced before. Figure 4 illustrates this edit restriction.

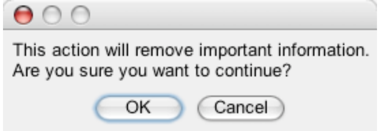
Safe Deletion of Rows. Another usual problem with non-normalized data is the deletion problem. Suppose in our running example that row 5 is deleted. In such a scenario, all the information about the `pg36` property is lost. However, it is likely that the user wanted to delete the rental transaction represented by that row only. In order to



Protected cells can not be modified.

Figure 4: In order to prevent update anomalies some columns must not be editable.

prevent this type of deletion problems, we have added a button per spreadsheet row (see Figure 2). When pressed, this button detects whether the end user is deleting important information included in the corresponding row. In case important information is removed by such deletion, a warning window is displayed, as shown in Figure 5.



This action will remove important information.
Are you sure you want to continue?

Figure 5: Window to warn the end user that crucial information may be deleted.

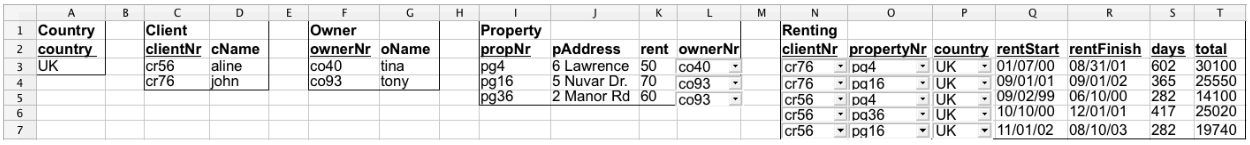
Apart from these new features, the user can still access traditional editing features, and can rely on recalculation of functional dependencies in the background.

Traditional Editing. Advanced programming language environments provide both advanced editing mechanisms and traditional ones (*i.e.*, text editing). In a similar way, a spreadsheet environment should allow the user to perform traditional spreadsheet editing, too. In traditional editing the end user is able to introduce data that may violate the relational database model that the spreadsheet data induces.

Recalculation of the Relational Database Model. Because standard editing allows the end user to introduce data violating the underlying relational model, we would like that the spreadsheet environment may enable/disable the safety features described in this section. When safety features are disabled, the end user would be able to introduce data that (temporarily) violates the (previously) inferred relational model. However, when the end user returns to safe editing, then the spreadsheet should infer a new relational model that will be used in future (safe) interactions.

2.2. *Refactored Spreadsheet Environment*

From the schema inferred we would also like to create a refactored spreadsheet which does not include any redundant information. Figure 6 presents such an optimized and modular spreadsheet for our running example. This spreadsheet is from now on termed *refactored*.



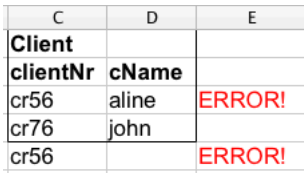
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	**Country**		**Client**			**Owner**			**Property**					**Renting**						
2	**country**		**clientNr**	**cName**		**ownerNr**	**oName**		**propNr**	**pAddress**	**rent**	**ownerNr**		**clientNr**	**propertyNr**	**country**	**rentStart**	**rentFinish**	**days**	**total**
3	UK		cr56	aline		co40	tina		pg4	6 Lawrence	50	co40		cr76	pg4	UK	01/07/00	08/31/01	602	30100
4			cr76	john		co93	tony		pg16	5 Nuvar Dr.	70	co93		cr76	pg16	UK	09/01/01	09/01/02	365	25550
5									pg36	2 Manor Rd	60	co93		cr56	pg4	UK	09/02/99	06/10/00	282	14100
6														cr56	pg36	UK	10/10/00	12/01/01	417	25020
7														cr56	pg16	UK	11/01/02	08/10/03	282	19740

Figure 6: The spreadsheet after applying the third normal form refactoring.

This new spreadsheet consists of five tables/modules, each one delimited by the empty column. In Figure 6, from left to right, we have a table for *countries*, another for *clients*, for *owners*, for *properties*, and for the *renting* action itself. Note that all these tables are in a single worksheet for presentation purposes, only. In a real spreadsheet environment, however, they would be organized in different worksheets.

As we explained before, the obtained modularity solves three well-known problems in databases, namely the insertion, modification and deletion anomalies. These problems do not exist in the generated spreadsheet because data is normalized.

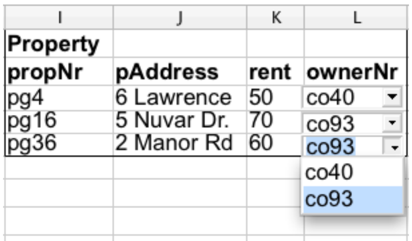
Key Columns. Moreover, we would like that the generated spreadsheet respected the schema. For example, in the *clients* table, the generated spreadsheet would not allow the user to introduce two clients with the same code, that is, the same *clientNr*. If that error occurs, the spreadsheet system should warn the user as shown in Figure 7. Obviously, it is not possible to perform this validation in the original spreadsheet.



C	D	E
Client		
clientNr	cName	
cr56	aline	ERROR!
cr76	john	
cr56		ERROR!

Figure 7: If the user introduces a new row in a table with a previously used $clientNr$ the spreadsheet will immediately produce an error.

Reference Columns. The reader may have noticed that, for example, column L (Figure 6) contains *combo boxes* on the cells. From the relational schema, we can see that this column, $ownerNr$, is a foreign key to the owner code in the *owners* table. In these cases, we would like that the new spreadsheet guarantees that the user only inserts values that already exist in the referenced table, since this is the definition of a foreign key. In fact, these columns should be locked for editing. This feature is illustrated in Figure 8.



I	J	K	L
Property			
propNr	pAddress	rent	ownerNr
pg4	6 Lawrence	50	co40
pg16	5 Nuvar Dr.	70	co93
pg36	2 Manor Rd	60	co93

Figure 8: Foreign key columns are filled in using combo boxes.

The refactored spreadsheet not only improves modularity and detects the introduction of incorrect data, but also eliminates redundancy: indeed, the redundancy present in the original spreadsheet has been eliminated. As expected, the names of the two clients occur only once.

In this section we have described an instance of our techniques. In fact, the spreadsheet programming environments were automatically produced from the original spreadsheet displayed in Figure 1. In the following sections we will present in detail the techniques to perform such an automatic spreadsheet refactoring.

3. From Spreadsheets to Relational Databases

This section explains how to extract functional dependencies from the spreadsheet data and how to construct a normalized relational database schema modeling such data. These techniques were introduced in detail in our work on defining a bidirectional mapping between spreadsheets and relational databases [7]. In this section we present an extension to that algorithm that uses spreadsheet specific properties in order to infer a more realistic set of functional dependencies. Before we present the algorithm and our new extension, let us briefly introduce a few database definitions that are needed to understand functional dependencies.

Relational Databases. A *relational schema* R is a finite set of attributes $\{A_1, ..., A_k\}$. Corresponding to each attribute A_i is a set D_i called the *domain* of A_i. These domains are arbitrary, non-empty sets, finite or countably infinite. A *relation* (or *table*) r on a relation schema R is a finite set of *tuples* (or *rows*) of the form $\{t_1, ..., t_k\}$. For each $t \in r$, $t(A_i)$ must be in D_i. A *relational database schema* is a collection of relation schemas $\{R_1, ..., R_n\}$. A *Relational Database* (RDB) is a collection of relations $\{r_1, ..., r_n\}$.

Each tuple is uniquely identified by a minimum non-empty set of attributes called a *Primary Key* (PK). On certain occasions there may be more than one set suitable for becoming the primary key. They are designated *candidate keys* and only one is chosen to become primary key. A *Foreign Key* (FK) is a set of attributes within one relation that matches the primary key of some relation (possibly the same).

The normalization of a database is important to prevent data redundancy. Although there are several different normal forms, in general, a RDB is considered normalized if it respects the *Third Normal Form* (3NF) [11].

Discovering Functional Dependencies. In order to define the RDB schema, we first need to compute the functional dependencies presented in a given spreadsheet data. In [7] we reused the well known data mining algorithm, named FUN, to infer such dependencies. This algorithm was developed in the context of databases with the main goal of inferring all existing functional dependencies in the input data. As a result, FUN may infer a large set of functional dependencies depending on the input data. For our example, we list the functional dependencies inferred from the data using FUN:

$$\begin{array}{ll}
\mathit{clientNr} & \rightharpoonup \mathit{cName}, \mathit{country} \\
\mathit{propNr} & \rightharpoonup \mathit{country}, \mathit{pAddress}, \mathit{rent}, \mathit{ownerNr}, \mathit{oName} \\
\mathit{cName} & \rightharpoonup \mathit{clientNr}, \mathit{country} \\
\mathit{pAddress} & \rightharpoonup \mathit{propNr}, \mathit{country}, \mathit{rent}, \mathit{ownerNr}, \mathit{oName} \\
\mathit{rent} & \rightharpoonup \mathit{propNr}, \mathit{country}, \mathit{pAddress}, \mathit{ownerNr}, \mathit{oName} \\
\mathit{ownerNr} & \rightharpoonup \mathit{country}, \mathit{oName} \\
\mathit{oName} & \rightharpoonup \mathit{country}, \mathit{ownerNr}
\end{array}$$

Note that, data contained in the spreadsheet exhibits all those dependencies. In fact, even the non-natural dependency $\mathit{rent} \rightharpoonup \mathit{propNr}, \mathit{country}, \mathit{pAddress}, \mathit{ownerNr}, \mathit{oName}$ is inferred. Indeed, the functional dependencies derived by the FUN algorithm depend heavily on the quantity and quality of the data. Thus, for small samples of data, or data that exhibits too many or too few dependencies, the FUN algorithm may not produce the desired functional dependencies.

Note also that the $\mathit{country}$ column occurs in most of the functional dependencies although only a single country actually appears in a column of the spreadsheet, namely UK. Such single value columns are common in spreadsheets. However, for the FUN algorithm they induce redundant fields and redundant functional dependencies.

In order to derive more realistic functional dependencies for spreadsheets we have extended the FUN algorithm so that it considers the following spreadsheet properties:

- *Single value columns*: these columns produce a single functional dependency with no right hand side ($\mathit{country} \rightharpoonup$, for example). These columns are not considered when finding other functional dependencies.
- *Semantic of labels*: we consider label names as strings and we look for the occurrence of words like *code, number, nr, id* given them more priority when considered as primary keys.
- *Column arrangement*: we give more priority to functional dependencies that respect the order of columns. For example, $\mathit{clientNr} \rightharpoonup \mathit{cName}$ has more priority than $\mathit{cName} \rightharpoonup \mathit{clientNr}$.

Moreover, to minimize the number of functional dependencies we consider the smallest subset that includes all attributes/columns in the original set computed by FUN. The result of our spreadsheet functional dependency inference algorithm is:

$$\begin{array}{ll}
\mathit{country} & \rightharpoonup \emptyset \\
\mathit{clientNr} & \rightharpoonup \mathit{cName} \\
\mathit{ownerNr} & \rightharpoonup \mathit{oName} \\
\mathit{propNr} & \rightharpoonup \mathit{pAddress}, \mathit{rent}, \mathit{ownerNr}, \mathit{oName}
\end{array}$$

This set of dependencies is very similar to the one presented in the previous section. The exception is the last functional dependency which has an extra attribute (oName).

An important property to guarantee in a set of relational schemas is the *lossless decomposition* property. *Lossless decomposition* means that if we decompose a relation into smaller relations, it is possible to undo the process and recover the original relation.

Moreover, it is possible that some columns of our spreadsheet are not included in any functional dependency.

To guarantee that they appear in the final schema and to ensure the lossless decomposition property we need to give an extra functional dependency to the algorithm. In fact, we use Maier's strategy [15]: we generate an additional functional dependency which contains all the columns of our spreadsheet as antecedent (the exception are the ones defined by formulas). The consequent of the dependency is a newly introduced attribute, in this case $newAttribute$, plus the attributes representing formulas (the relation between formulas and functional dependencies is explained in the following paragraphs). Note that this new attribute is just a "phantom" attribute required by the algorithm and can be removed when the final set of functional dependencies is created [15]. For our example such functional dependency is as follows:

$$\begin{array}{l} country, clientNr, cName, ownerNr, oName, propNr, pAddress, rent, rentStart, rentFinish \\ \quad \rightharpoonup days, total, newAttribute \end{array}$$

Thus, the final set of functional dependencies is as follows:

$$\begin{array}{ll} country & \rightharpoonup \emptyset \\ clientNr & \rightharpoonup cName \\ ownerNr & \rightharpoonup oName \\ propNr & \rightharpoonup pAddress, rent, ownerNr, oName \\ \multicolumn{2}{l}{country, clientNr, cName, ownerNr, oName, propNr, pAddress, rent, rentStart, rentFinish} \\ \multicolumn{2}{l}{\quad \rightharpoonup days, total, newAttribute} \end{array}$$

Spreadsheet Formulas. Spreadsheet systems differ from database systems by the use of formulas. Spreadsheet formulas define the values of some elements in terms of other elements. As a consequence, a formula explicitly defines a functional dependency. Such functional dependencies need to be included in the induced functional dependencies of our algorithm. Let us consider again the house rental spreadsheet, in which column $days$ is computed by subtracting the column $rentFinish$ from $rentStart$: this is usually written as follows `H3 = G3 - F3`. This formula states that the values of `G3` and `F3` determine the value of `H3`, thus inducing the following functional dependency: $rentStart, rentFinish \rightharpoonup days$.

Formulas can have references to other formulas. Consider, for example, the second formula of the running example `J3 = H3 * I3`, which defines the total rent by multiplying the total number of days by the value of the rent. Because `H3` is defined by another formula, the values that determine `H3` also determine `J3`. As a result, the two formulas in our example induce the following functional dependencies:

$$\begin{array}{ll} rentStart, rentFinish & \rightharpoonup days \\ rentStart, rentFinish, rent & \rightharpoonup total \end{array}$$

In general, a spreadsheet formula of the following form $X_0 = f(X_1, \ldots, X_n)$ induces the following functional dependency: $X_1, \ldots, X_n \rightharpoonup X_0$. In spreadsheet systems, formulas are usually introduced by copying them through all the elements in a column, thus making the functional dependency explicit in all the elements. This may not always be the case and some elements can be defined otherwise (e.g. by using a constant value or a different formula). In both cases, all the cells referenced must be used in the antecedent of the functional dependency.

These functional dependencies are useful for the mapping of spreadsheets to databases as presented in [7]. Moreover, they are also used to create the final relational schema since they are necessary to construct the functional dependency that guarantees the lossless property.

Normalizing Functional Dependencies. Having computed the functional dependencies, we can now normalize them. Next, we show the results produced by the *synthesize* algorithm introduced by Maier in [15]. The *synthesize* algorithm receives a set of functional dependencies as argument and returns a new set of *compound functional dependencies*. A *compound functional dependency* (CFD) has the form $(X_1, \ldots, X_n) \rightharpoonup Y$, where $X_1, \ldots, X_n$ are all distinct subsets of a scheme R and Y is also a subset of R. A relation r satisfies the CFD $(X_1, \ldots, X_n) \rightharpoonup Y$ if it satisfies the functional dependencies $X_i \rightharpoonup X_j$ and $X_i \rightharpoonup Y$, for all $1 \leqslant i, j \leqslant n$. In a CFD, $(X_1, \ldots, X_n)$ is the *left side*, $X_1, \ldots, X_n$ are the *left sets* and Y is the *right side*. For more details about CFDs please refer to [15]. Note that before applying this algorithm, all the functional dependencies with antecedents' attributes representing formulas should be eliminated since a primary key must not change over time.

Next, we list the compound functional dependencies computed from the functional dependencies induced by our running example.

$$\begin{array}{ll}
(\{\mathit{country}\}) & \rightharpoonup \emptyset \\
(\{\mathit{clientNr}\}) & \rightharpoonup \{\mathit{cName}\} \\
(\{\mathit{ownerNr}\}) & \rightharpoonup \{\mathit{oName}\} \\
(\{\mathit{propNr}\}) & \rightharpoonup \{\mathit{pAddress}, \mathit{rent}, \mathit{ownerNr}\} \\
\multicolumn{2}{l}{(\{\mathit{country}, \mathit{clientNr}, \mathit{propNr}, \mathit{rentStart}, \mathit{rentFinish}\}) \rightharpoonup \{\mathit{days}, \mathit{total}, \mathit{newAttribute}\}}
\end{array}$$

Note that the redundant attributes have been removed. This derives directly from the normalization process introduced by Maier [15] where both redundant dependencies and redundant attributes are removed, that is, dependencies and attributes that do not add new information to the model are discarded. More details about this algorithm can be found in [15].

Computing the Relational Database Schema. Each compound functional dependency may define several candidate keys for each table. However, to fully characterize the relational database schema we need to choose the primary key from those candidates. To find such keys we use a simple algorithm: we produce all the possible tables using each candidate key as the primary key. In general, and since the set of functional dependencies is already reduced to its minimum, the sets of candidate keys are small (in many cases, as in the example, with only one element). Thus, not many combinations are generated. Finally, we apply again the same heuristics used to choose the functional dependencies to select the best tables from all the generated ones. For instance, we give more priority to tables that respect the order of columns than to others. The same happens to the other heuristics. In fact, we are just applying the same algorithm to choose the best functional dependencies to work with as before.

$$\begin{array}{l}
\underline{\mathit{country}} \\
\underline{\mathit{clientNr}}, \mathit{cName} \\
\underline{\mathit{ownerNr}}, \mathit{oName} \\
\underline{\mathit{propNr}}, \mathit{pAddress}, \mathit{rent}, \#\mathit{ownerNr} \\
\underline{\#\mathit{country}, \#\mathit{clientNr}, \#\mathit{propNr}, \mathit{rentStart}, \mathit{rentFinish}}\ \mathit{days}, \mathit{total}
\end{array}$$

This relational database model corresponds exactly to the one shown in Section 2. Note that the synthesize algorithm removed the redundant attribute oName that occurred in the last functional dependency through the normalization process. Note also that we have removed the "phantom" attribute ($\mathit{newAttribute}$) from the final model since it is no longer useful and is not part of the spreadsheet.

This set of functional dependencies completely characterizes our spreadsheet. Indeed to construct the *refactored* spreadsheet all these dependencies are necessary. However, to construct the *schema-aware* spreadsheet, the last dependency is not necessary. Remember that it exists only to guarantee the lossless property, necessary when the data is rearrange and if one wants to have the data in its initial format later. Since in the *schema-aware* spreadsheet we do not rearrange the data, it is not necessary, and in fact, it would only duplicate some of the visual objects and formulas, not adding any new functionality.

More details about the system here presented to infer functional dependencies and compute a relational database schema from them can be found in [16].

4. Building Spreadsheet Programming Environments

This section presents techniques to refactor spreadsheets into powerful spreadsheet programming environments as described in Section 2. This spreadsheet refactoring is implemented as the embedding of the inferred functional dependencies and the computed relational model in the spreadsheet. This embedding is modeled in the spreadsheet itself by standard formulas and visual objects: formulas are added to the spreadsheet to guide end users to introduce correct data.

Before we present how this embedding is defined, let us first define a spreadsheet. A spreadsheet can be seen as a partial function $S : A \rightarrow V$ mapping addresses to spreadsheet expressions. Elements of S are called *cells* and are represented as (a, e). A cell address is taken from the set $A = \mathbb{N} \times \mathbb{N}$. An expression $e \in E$ can be an input plain value $c \in C$ like a string or a number, a reference to other cells using addresses $a \in A$, or a function name $f \in F$ that can be applied to one or more expressions: $e \in E ::= c \mid a \mid f(e, \ldots, e)$.

4.1. Schema-aware Spreadsheet Environment

Let us start by defining the generation of the *schema-aware* spreadsheet. First, we show in detail how each computed relational table is used to provide an auto-completion mechanism in the spreadsheet environment. Next, we use the attributes of each relational table to include a non-editable mechanism on the programming environment, which will guide end users to avoid update inconsistencies. Next, we will show how to embed a mechanism that avoids the deletion of unintended data.

Auto-completion of Column Values. This feature is implemented by embedding each of the relational tables in the spreadsheet. It is implemented by a spreadsheet formula and a combo box visual object. The combo box displays the possible values of one column, associated to the primary key of the table, while the formula is used to fill in the values of the columns that the primary key determines.

Let us consider the table $\underline{ownerNr}, oName$[1] from our running example. In the spreadsheet, $ownerNr$ is in column K and $oName$ in column L. This table is embedded in the spreadsheet introducing a combo box containing the existing values in the column K (as displayed in Figure 2). Knowing the value in the column K we can automatically introduce the value in column L. To achieve this, we embed the following formula in row 7 of column L:

$$S\ (L, 7) = \textbf{if}\ (\textbf{isna}\ (\textbf{vlookup}\ (K7, K2 : L6, 2, 0)), \texttt{""}, \textbf{vlookup}\ (K7, K2 : L6, 2, 0))$$

This formula uses a (library) function **isna** to test if there is a value in column K. In case that value exists, it searches (with the function **vlookup**) the corresponding value in the column L and references it. The formula **vlookup** receives the value to look for (or a reference to it), a range with two or more columns with data to be searched, the index of the column within the search range from where the result must be obtained, and a parameter that indicates whether the first column of the range is sorted in ascending order. If there is no selected value, it produces the empty string. The combination of the combo box and this formula guides the user to introduce correct data as illustrated in Figure 2.

We have just presented a particular case of the formula and visual object induced by a relational table. Next we present the general case. Let min_r be the very next row after the existing data in the spreadsheet, max_r the last row in the spreadsheet, and r_1 the first row with already existing data. Each relational database table $\underline{a_1}, ..., \underline{a_n}, c_1, ..., c_m$, with $a_1, ..., a_n, c_1, ..., c_m$ column indexes of the spreadsheet, induces firstly, a combo box defined as follows:

$$\begin{aligned}
&\forall\ c\ \in\ \{a_1, ..., a_n\}, \forall\ r\ \in\ \{min_r, ..., max_r\} : \\
&\quad S\ (c, r) = combobox := \{\, linked\_cell\ \ := (c, r); \\
&\qquad\qquad\qquad\qquad\qquad\quad source\_cells := (c, r_1) : (c, max_r)\,\}
\end{aligned}$$

secondly, a spreadsheet formula defined as:

$$\begin{aligned}
&\forall\ c\ \in\ \{c_1, ..., c_m\}, \forall\ r\ \in\ \{min_r, ..., max_r\} : \\
&\quad S\ (c, r) = \textbf{if}\ (\textbf{if}\ (\textbf{isna}\ (\textbf{vlookup}\ ((a_1, r), (a_1, r_1) : (c, r-1), r - a_1 + 1, 0)), \\
&\qquad\qquad\qquad\quad \texttt{""}, \\
&\qquad\qquad\qquad\quad \textbf{vlookup}\ ((a_1, r), (a_1, r_1) : (c, r-1), r - a_1 + 1, 0)) \\
&\qquad\qquad\qquad == \\
&\qquad\qquad\quad \textbf{if}\ (\textbf{isna}\ (\textbf{vlookup}\ ((a_2, r), (a_2, r_1) : (c, r-1), r - a_2 + 1, 0)), \\
&\qquad\qquad\qquad\quad \texttt{""}, \\
&\qquad\qquad\qquad\quad \textbf{vlookup}\ ((a_2, r), (a_2, r_1) : (c, r-1), r - a_2 + 1, 0)) \\
&\qquad\qquad\qquad == \\
&\qquad\qquad\qquad \ldots
\end{aligned}$$

[1] Remember that underlined attributes are the primary key of the table.

```
$==$
$\textbf{if}\ (\textbf{isna}\ (\textbf{vlookup}\ ((a_n, r), (a_n, r_1) : (c, r-1), r - a_n + 1, 0)),$
    $"\,"$,
    $\textbf{vlookup}\ ((a_n, r), (a_n, r_1) : (c, r-1), r - a_n + 1, 0)),$
$\textbf{vlookup}\ ((a_1, r), (a_1, r_1) : (c, r-1), r - a_1 + 1, 0),$
$"\,")$
```

where $==$ represents the mathematical equality test, which we have implemented as follows: $a == b == c$ is defined in Excel as **and** $(a = b, a = c)$. This formula must be used for each non primary key column created by our algorithm. Intuitively, this formula checks, for a given non primary key cell, if all the values chosen by the user from the primary key columns in the same row of that cell, agree on the value of the non primary key column. If so, then such a value is automatically inserted in the cell under consideration.

Let us consider the functional dependency $country, ownerNr \rightharpoonup propNr$ in our example. Note that this is a functional dependency that holds but is not used. Each cell of the column $propNr$ receives an instance of the given formula. In this case the formula will have two inner **if**, one to check $country$ and another to check $ownerNr$. When in one row there are values chosen for $country$ and $ownerNr$, if the **vlookup** checking these two columns finds the corresponding $propNr$ value to be the same, then such a value is shown in the $propNr$ cell.

In general, each conditional **if** inside the main **if** is responsible for checking a primary key column. In the case a primary key column value is chosen, so **isna** (**vlookup** (...)) is true, the formula calculates the corresponding non primary key column value, **vlookup** (...). If the values chosen by all primary key columns are the same (represented by all the equality tests $==$), then that value is used in the non primary key column. This formula considers tables with primary keys consisting of multiple attributes (columns). Note also that the formula is defined in each column associated to non-key attribute values.

The example table analyzed before is an instance of this general one. In the table $\underline{ownerNr}$, $oName$, $ownerNr$ is a_1, $oName$ is c_1, c is L, r_1 is 2, min_r is 7. The value of max_r is always the last row supported by the spreadsheet system.

Foreign keys pointing to primary keys become very helpful in this setting. For example, if we have the relational tables $\underline{A}, B$ and $\underline{B}, C$ where B is a foreign key from the second table to the first one, then when we perform auto-completion in column A, both B and C are automatically filled in. This was the case presented in Figure 2.

Non-Editable Columns. To prevent wrong introduction of data, and thus, producing update anomalies, we protect some columns from random modification. Suppose in our example the user intends to insert a new entry registering the renting of property `pg36`. If the system would let to insert again the property key, then by a mistake, the user could insert a different key for the same property, thus inputing inconsistent data. A similar situation could happen to non-key columns. Thus, a relational table, such as $\underline{a_1}, ..., \underline{a_n}, c_1, ..., c_m$, induces the non-editability of columns $a_1, ..., a_n, c_1, ..., c_m$ in an arbitrary way. That is to say that all columns that form a table become non-editable, unless the user explores the system features. For non-key columns this means to use the auto-completion feature. That is, non-key columns are edited when key column change since the key value automatically fill in the non-key ones. On the other hand, key columns are edited when the user selects values from the existing combo boxes. Figure 4 illustrates such a restriction. In the case where the end user really needs to change the value of such protected columns, we provide traditional editing (explained below).

Safe Deletion of Rows. Another typical problem with non-normalized data is the deletion of data. Suppose in our running example that row 5 is deleted. All the information about property `pg36` is lost, although the user would probably want to delete that rental transaction only. To correctly delete rows in the spreadsheet, a button is added to each row in the spreadsheet as follows: for each relational table $\underline{a_1, ..., a_n}, c_1, ..., c_m$ each button checks, on its corresponding row, the columns that are part of the primary key, $a_1, ..., a_n$. For each primary key column, it verifies whether the value to remove is the last one.

Let $c \in \{a_1, ..., a_n\}$, let r be the row where the button is positioned, r_1 be the first row of column c with data and r_n be the last row of column c with data. The test is defined as follows:

$$\textbf{if}\ (\textbf{isLast}\ ((c, r), (c, r_1) : (c, r_n)), \textbf{showMessage}, \textbf{deleteRow}\ (r))$$

If the value is the last one of its kind, the spreadsheet warns the user (**showMessage**) as can be seen in Figure 5. If the user presses the `OK` button, the spreadsheet will remove the row. In the other case, `Cancel`, no action will be performed. In the case the value is not the last one, the row will simply be removed, **deleteRow** (r).

More specifically, **isLast** verifies if the value in the cell passed as argument is the last one of its kind, that is, if in the same column there is no other value like it. For instance, suppose in our example row 2 is deleted. Then, the value `cr76` in cell A3 would be the last one of its kind since there is no other cell with the same value in that column. The function **showMessage** simply shows the alert message to the user. Finally, **deleteRow** deletes the row that it receives as argument. All these functions are implemented in OpenOffice Basic and their code is omitted since it would not improve the understandability of the work.

This mechanism will be triggered, for instance, if the user tries to delete row 5 of our running example, since in column $propNr$ this row contains the last data about the house with code `pg36`.

Traditional Editing. Advanced programming language environments provide both advanced editing mechanisms and traditional ones (*i.e.*, text editing). In a similar way, a spreadsheet environment should allow the user to perform traditional spreadsheet editing too. Thus, the environment should provide a mechanism to enable/disable the safety features described in this section. When safety features are disabled, the end user may introduce data that violates the (previously) inferred relational model. However, when the end user returns to safe editing, the spreadsheet infers a new relational model that will be used in future (safe) interactions.

4.2. The Refactored Spreadsheet Environment

In this section we explain how to generate the *refactored* spreadsheet.

Key Columns. The keys of the functional dependencies are represented in the spreadsheet by being underlined. For instance, the column labeled *clientNr* is underlined and this is the key of the client's table. As in relational databases there cannot be two rows with the same key. If such a situation occurs, the spreadsheet will warn the user as shown in Figure 7.

The column immediately after each table contains a formula that computes whether there are two equal values in the key columns. For clients the formula is shown next:

$$= \textbf{if}\ (\textbf{and}\ (\$c\$3 : \$c\$6501 <> \texttt{""}; \textbf{countif}\ (\$c\$3 : \$c\$6501; \$c\$3 : \$c\$6501) > 1); \texttt{"ERROR!"}; \texttt{""})$$

Assuming a_1 is the reference to the first cell with data of the key column and a_n is the reference to the last, generically this formula is constructed as follows:

$$= \textbf{if}\ (\textbf{and}\ (a_1 : a_n <> \texttt{""}; \textbf{countif}\ (a_1 : a_n; a_1 : a_n) > 1; \texttt{"ERROR!"}; \texttt{""})$$

The *if* condition calculates if there are repeated values. If so, the string `"ERROR!"` is shown, otherwise, the empty string populates the cell.

Reference Columns. The second feature this *refactored* environment provides is safe references to other tables. In the relational database realm these are known as *foreign keys*. Since we have this information from the relational schema calculated before, we can integrate it in the spreadsheet. For instance, the table containing the properties has a reference to the property's owner, in column L. As shown in Figure 8 such column contains *combo boxes* in the cells. Thus, these cells can only contain owners that in fact exist in the spreadsheet.

To accomplish this, each relational database table $\underline{\#a_m}, ..., \underline{\#a_n}, \underline{a_{n+1}}, ..., \underline{a_r}, \#b_s, ..., \#b_t, ..., b_{t+1}, ..., b_u$ that references another table $\underline{c_m}, ..., \underline{c_n}, \underline{c_s}, ..., \underline{c_t}, d_1, ..., d_k$, where a_*, b_*, c_*, d_* are column indexes of the spreadsheet, induces a combo box defined as follows:

$$\begin{aligned}
&\forall\ (i_x)\ \in\ \{\, a_m, ..., a_n \,\}\ \cup\ \{\, b_s, ..., b_t \,\} : \\
&\quad S\ (i_x, r) = \mathit{combobox} := \{\, \mathit{linked\_cell} := (i_x, r); \\
&\qquad\qquad\qquad\qquad\qquad \mathit{source\_cells} := (c_x, \mathit{min}_r) : (c_x, \mathit{max}_r)\}
\end{aligned}$$

where min_r is the very first row with data in the spreadsheet, max_r the last row in the spreadsheet. This means that each column that is a reference to another column will have a combo box with the values available in the referenced column.

4.3. HaExcel Add-in

We have implemented the FUN algorithm, the extensions described in this paper, the *synthesize* algorithm, and the embedding of the relational model in the HASKELL programming language [17]. We have also defined the mapping from spreadsheet to relational databases in the same framework named HaExcel [7]. Finally, we have extended this framework to produce the visual objects and formulas to model the relational tables in the spreadsheet. An Excel add-in has been also constructed so that end users can use spreadsheets in this popular system and at the same time our safety features.

5. Applicability Experiment

In this section we present an experiment performed to evaluate the applicability of our approach. Indeed we want to answer a single question: *can this approach be applied to a wide range of spreadsheets*? If this is not possible, then the approach needs more work. On the other hand, if this can be applied widely to spreadsheets, then we have a starting point.

For this we have run an experiment using the EUSES Corpus [10]. This corpus was conceived as a shared resource to support research on technologies for improving the dependability of spreadsheet programming. It contains more than 4500 spreadsheets gathered from different sources and developed for different domains. These spreadsheets are assigned to eleven different categories including financial (containing 19% of the total number of spreadsheets), inventory (17%), homework (14%), grades (15%), database (17%) and modeling (17%) (the remaining 1% represents other spreadsheets). Among the spreadsheets in the corpus, about 4.4% contain macros, about 2.3% contain charts, and about 56% do not have formulas being only used to store data. This corpus is widely accepted by the research community and has been used in other works [18, 19, 20].

In this experiment we have selected the first ten spreadsheets from each of the eleven categories of the corpus. We then applied our tool to each spreadsheet, obtaining different results (see also Table 1): a few spreadsheets failed to parse, due to glitches in the Excel to Gnumeric conversion (which we use to bring spreadsheets into a processable form). Other spreadsheets were parsed, but no tables could be recognized in them, *i.e.*, their users did not adhere to any of the supported layout conventions. The layout conventions we support are the ones presented in the UCheck project [21], for instance, it is not possible to work with spreadsheets that have only sparse filled cells not forming a table or tables, neither with spreadsheets that do not label each table in the first row or column. This was the case for about half of the spreadsheets in our experiment. The other spreadsheets were parsed, tables were recognized, and edit assistance was generated for them. We will focus on the last groups in the upcoming sections.

Processed Spreadsheets. The results of processing our sample of spreadsheets from the EUSES corpus are summarized in Table 1. The rows of the table are grouped by category as documented in the corpus. The first three columns contain size metrics on the spreadsheets. They indicate how many tables were recognized, how many columns are present in these tables, and how many cells. For example, the first spreadsheet in the *financial* category contains 15 tables with a total of 65 columns and 242 cells.

Table 1: Results of processing the selected spreadsheets.

File name	Recognized tables	Columns	Cells	FDs	Columns with safe insertion & deletion	Auto-completion columns	Non-editable columns
cs101							
Act4_023_capen	5	24	402	0	0	0	0
act3_23_bartholomew	6	21	84	1	8	1	9
act4_023_bartholomew	6	23	365	0	0	0	0
meyer_Q1	2	8	74	0	0	0	0
posey_Q1	5	23	72	0	8	0	8

continues on the next page

Table 1 – continuation of previous page

File name	Recognized tables	Columns	Cells	FDs	Columns with safe insertion & deletion	Auto-completion columns	Non-editable columns
database							
%5CDepartmental%20Fol#A8…	2	4	3463	0	0	0	0
00061r0P802-15_TG2-Un…69	23	55	491	0	18	4	21
00061r5P802-15_TG2-Un…6C	30	83	600	25	21	5	26
0104TexasNutrientdb	5	7	77	1	1	1	2
01BTS_framework	52	80	305	4	23	2	25
03-1-report-annex-5	20	150	1599	12	15	8	22
filby							
BROWN	5	14	9047	2	3	1	4
CHOFAS	6	48	4288	3	3	1	4
financial							
03PFMJOURnalBOOKSFina...	15	65	242	0	7	0	7
10-formc	12	20	53	8	5	4	9
forms/3							
ELECLAB3.reichwja.xl97	1	4	44	0	0	0	0
burnett-clockAsPieChart	3	8	14	0	1	0	1
chen-heapSortTimes	1	2	24	0	0	0	0
chen-insertSortTimes	1	2	22	0	0	0	0
chen-lcsTimes	1	2	22	0	0	0	0
chen-quickSortTimes	1	2	24	0	0	0	0
cs515_npeg_chart.reichwja.xl97	7	9	93	0	0	0	0
cs515_polynomials.reichwja.xl97	6	12	105	0	0	0	0
cs515_runtimeData.reichwja.X...	2	6	45	0	0	0	0
grades							
0304deptcal	11	41	383	19	18	17	28
03_04ballots1	4	20	96	6	4	0	4
030902	5	20	110	0	0	0	0
031001	5	20	110	0	0	0	0
031501	5	15	51	31	3	1	4
homework							
01_Intro_Chapter_Home#A9171	6	15	2115	0	1	0	1
01readsdis	4	16	953	5	4	3	6
02%20fbb%20medshor	1	7	51	0	0	0	0
022timeline4dev	28	28	28	0	0	0	0
026timeline4dev	28	28	30	0	2	0	2
03_Stochastic_Systems#A9172	4	6	48	0	2	0	2
04-05_proviso_list	79	232	2992	0	25	0	25
inventory							
02MDE_framework	50	83	207	10	31	1	32
02f202assignment%234soln	37	72	246	7	20	1	21
03-1-report-annex-2	5	31	111	10	5	5	8
03singapore_elec_gene#A8236	9	45	153	3	5	2	7
0038	10	22	370	0	0	0	0

continues on the next page

Table 1 – continuation of previous page

File name	Recognized tables	Columns	Cells	FDs	Columns with safe insertion & deletion	Auto-completion columns	Non-editable columns
modeling							
%7B94402d63-cdd8-4cc3#A...	1	3	561	0	0	0	0
%EC%86%90%ED%97%8C...	1	10	270	13	7	5	9
%EC%9D%98%EB%8C%80...	1	7	1442	4	4	5	6
%EC%A1%B0%EC%9B%90...	2	17	534	18	13	5	15
%ED%99%98%EA%B2%B...	3	7	289	2	1	2	3
0,10900,0-0-45-109057-0,00	4	14	6558	9	9	2	10
00-323r2	24	55	269	31	9	6	15
00000r6xP802-15_Docum#A...	3	13	3528	10	9	3	11
003_4	25	50	2090	0	0	0	0

The fourth column shows how many functional dependencies were extracted from the recognized tables. These are the non-trivial functional dependencies that remain after we use our extension to the FUN algorithm to discard redundant dependencies. The last three columns are metrics on the generated edit assistance. In some cases, no edit assistance was generated, indicated by zeros in these columns. This situation occurs when no (non-trivial) functional dependencies are extracted from the recognized tables. In the other cases, the three columns respectively indicate:

- For how many columns a combo box has been generated for *controlled insertion*. The same columns are also enhanced with the *safe deletion of rows* feature.
- For how many columns the *auto-completion of column values* has been activated, *i.e.*, for how many columns the user is no longer required to insert values manually.
- How many columns are locked to prevent edit actions where information that does not appear elsewhere is deleted inadvertently.

For example, for the first spreadsheet of the *inventory* category, combo boxes have been generated for 31 columns, auto-completion has been activated for 1 column, and locking has been applied to 32 columns. Note that for the categories *jackson* and *personal*, no results were obtained due to absent or unrecognized layout conventions or to the size of the spreadsheets (more than 150,000 cells).

Observations. On the basis of these results, a number of interesting observations can be made. For some categories, edit assistance is successfully added to almost all spreadsheets (e.g. *inventory* and *database*), while for others almost none of the spreadsheets lead to results (e.g. the *forms/3* category). The latter may be due to the small sizes of the spreadsheets in this category. For the *financial* category, we can observe that in only 2 out of 10 sample spreadsheets tables were recognized, but edit assistance was successfully generated for both of these.

The *percentage* of columns for which edit assistance was generated varies. The highest percentage was obtained for the second spreadsheet of the *modeling* category, with 9 out of 10 columns (90%). A good result is also obtained for the first spreadsheet of the *grades* category with 28 out of 41 columns (68.3%). On the other hand, the 5^{th} of the *homework* category gets edit assistance for only 2 out of 28 columns (7.1%). The number of columns with combo boxes often outnumbers the columns with auto-completion. This may be due to the fact that many of the functional dependencies are small, with many having only one column in the antecedent and none in consequent.

In summary, from 102 selected spreadsheets (although there are 11 categories, *personal* only has 5 spreadsheets and *cs101* only 7) we could automatically process 50 spreadsheets. From these 50 spreadsheets, we could automatically generate assistance for 33 (66%). Thus, we could generate in an automatic fashion an environment for more than

32% of all spreadsheets from the 11 different categories. This percentage could be improved if our parsing system for spreadsheets was better. In fact, considering only the ones we could parse, our technique could create assistance for 66% of such spreadsheets.

Discussion. Our experiment justifies two conclusions. Firstly, the tool is able to successfully add edit assistance to a series of non-trivial spreadsheets. A more thorough study of these and other cases can now be done to identify technical improvements that can be made to the algorithms for table recognition and functional dependency extraction. For instance, for the categories *jackson* and *personal* we could not use any spreadsheets, and thus these are spreadsheets that we can study to improve our algorithms. Secondly, in the enhanced spreadsheets a large number of columns are generally affected by the generated edit assistance, which indicates that the user experience can be impacted in a significant manner. Thus, a validation experiment can be started to evaluate how users experience the additional assistance and to which extent their productivity and effectiveness can be improved. In fact we present in the next section such a study.

As a final remark we note that it is not possible to make any judgment about the quality of the generated environments. This would require another study, with real users and their spreadsheets. After applying our techniques to their spreadsheets users would need to evaluate the results, based on the benefits of such results. This would also require to train such users to work with these new spreadsheets. Unfortunately, such a study is outside of the scope of this paper. Nevertheless, in another study presented in [22] we have used the same technique here presented to infer functional dependencies from spreadsheets and used them to automatically create models to those spreadsheets. In that work we have shown that more than 76% of all models generated are *good* models (in a scale with the grades *bad*, *acceptable*, and *good*). Although this cannot be used to prove that the environments we here generate are of the same quality, since we are using the same technique to infer the structure and the functional dependencies that are the same for both works, we believe this gives a good hint about the generated environments.

6. Empirical Validation

In this section, we present an empirical study that we have conducted with the aim of analyzing the influence of using models in end-user spreadsheet productivity. In this study we consider both model-based spreadsheets proposed in this paper. We will refer to the refactored spreadsheet as *refactored* and to the other as *schema-aware*, as apposed to an *original* spreadsheet (without any model).

We assess end-user productivity in introducing, updating and querying data in those two model-based spreadsheets and in a traditional one. As the models we consider representative database-like spreadsheets. It should be clear that we are not analyzing all possible (types of) spreadsheets. Nevertheless, even considering spreadsheets strongly related to databases, the domain of our tests is clearly the spreadsheet environment.

In this paper we wish to answer the following research questions:

RQ1 Do end users introduce fewer errors when they use one of the model-based spreadsheets versus the *original* unmodified one?

RQ2 Does a particular model-based spreadsheet lead to fewer errors in particular tasks when compared to the *original* one?

RQ3 Are end users more efficient using the model-based spreadsheets than using the *original* one?

6.1. Study Design

As suggested in [23] we have divided this study in five different phases. Next we briefly summarize each of them so the reader can better understand the remaining of this section.

1 - Formulating hypothesis to test. After several discussions and meetings we finally formulated the hypothesis presented in this work: model-based spreadsheets can help end users committing less errors when editing and querying spreadsheets. More details in Section 6.1.4.

2- Observing a situation. Once we got enough and appropriate qualified participants we ran the study itself. In this case we selected 38 participants that were not studying any majors related to computer science or informatics (see Section 6.1.2). During the study, we screen casted the participants' computers and afterwards we collected the spreadsheets they worked on.

3 - Abstracting observations into data. We computed a series of statistics, that we present in detail in Section 6.2, over the spreadsheets participants developed during the study: we graded their performance and measured the time they took to perform the proposed tasks. All the data we used is available at the *SSaaPP* project web page `http://ssaapp.di.uminho.pt`. The tasks and the spreadsheets participants received are also available.

4 - Analyzing the data. The enormous collection of data that we gathered was later systematically analyzed. Each participant worked on three different spreadsheets and these three spreadsheets required the verification of 376 cells to assess the participant's answers (this was done mostly automatically). This analysis is presented in more detail in Section 6.2.

5 - Drawing conclusions with respect to the tested hypothesis. Based on the results we obtained, we finally drew some conclusions. We were also able to suggest some future research paths based on our work, which are presented in the Section 6.4.

Our study aimed to answer whether participants were able to perform their tasks with more accuracy and/or faster given the experimental environments. We used a within-subjects design[2], where each participant received 3 spreadsheets, one for each problem (DISHES, PROJECTS, PROPERTIES). Each of the 3 spreadsheets was randomly distributed under one of the 3 model (*original*, *refactored*, *schema-aware*). Participants were asked to do various tasks in each spreadsheet: data entry, editing, and calculations. They were encouraged to work as quickly as possible, but were not given time limits.

6.1.1. Methodology

Participants started the study by filling out a background questionnaire so we could collect their area of study and previous experience with spreadsheets, other programming languages and English comfort (Portuguese is their native language). An introduction to the study was given orally in English, this was explicitly not a tutorial for the different environments because the goal was to see if even without any introduction to the various models the participants would still be able to understand and complete the tasks. Since the order of the spreadsheets was randomized, they were told that the other sitting around them might appear to be moving faster, but that some tasks were shorter than others. After 2 hours participants were stopped if they were not already finished. Following the tasks they had a post session questionnaire which contained questions assessing their understanding of the different models, (3 questions for *refactored* and 4 for *schema-aware*). Correct answers could only be given by participants having understood the running models. Grading the questionnaires was done as follows: a correct answer receives total points; an incorrect answer receives 0 points and an answer that is not incorrect nor (totally) correct receives half of the points. We recorded the users' screens using screen capture technology. At the end of the study the user-completed spreadsheets were saved and graded for later analysis.

6.1.2. Participants

Recruitment was conducted through a general email message to the university, asking for students with spreadsheet experience and comfort with English. Of the hundreds that responded (there was a compensation involved), participants were selected based on spreadsheet experience, comfort with English, and majors outside of computer science and engineering. In total, 38 participants finished the study with data we were able to use (25 females, 11 males, and 2 who did not answer about their gender). Two participants did not try to solve one of the proposed tasks; for these participants, we included in the study only the tasks they undertook. A few participants' machines crashed and therefore they were eliminated from the study. The majority of participants were between 20-29 years of age, with

[2] A within-subjects design means that all treatments (in our case, the three different models) were applied to all participants, as opposed to have each treatment applied to different set of participants (in our case this would have mean to have three different sets of participants, each of which using a different model).

the remaining under 20. All were students at the university. About 2/3 were working on their Bachelor's degree, the remaining on their Masters. None were studying computer science or engineering and the most represented majors were medicine, economics, nursing and biology. A variety that is good for representing the end-user population of spreadsheets.

6.1.3. Tasks

The task lists were designed to include tasks that are known to be problematic in spreadsheets, which involve data insertion, edition and the use of formulas. The tasks were 1) add new information to the spreadsheet, 2) edit existing data in the spreadsheets and 3) do some calculations using the data in the spreadsheets.

Some of the tasks asked users to add many new rows of data, with the aim of a repetitive task being common in real-world situations. As we were designing the tasks, we imagined a type of data entry office scenario, where an office worker might receive on paper data which was initially filled out on a paper form and needed to be entered into a spreadsheet. This first task of data entry, in theory, should be fastest (and done with fewest entry errors) in the *refactored* spreadsheet. The second task, of making changes to existing data in a spreadsheet should also be easier within a *refactored* spreadsheet, since the change only needs to be made in one location, and therefore there would be less chance of forgetting to change it. The final task was to do some calculations using the data in the spreadsheet, such as averages, etc. This task was added because of the frequency of problems with formulas.

One of the spreadsheets used in the study, PROPERTIES, stores information about a house renting system (adapted from [11]). This spreadsheet has information about renters, houses and their owners as well as the dates and prices of the rents.

A second spreadsheet, DISHES, contains information about sales of detergents to dish washers. Information about the detergents, prices and the stores where they are sold is present on this spreadsheets (adapted from [24]).

The last spreadsheet, PROJECTS, stores information about projects, like the manager and delivery date, employees, and the instruments used buy them (adapted from [25]).

In the task list for DISHES, 67% (39 out of 58 cells needed to be changed) of the tasks consist of inserting new data, 21% (12/58) are editing tasks and 12% (7/58) involve calculations over the data in the spreadsheet. In the task list for PROJECTS, 80% (221/277) of the tasks are for inserting new data, 7% (20/277) for edition and 13% (36/227) for calculations. Finally, for PROPERTIES, inserting data tasks are 56% (64/115) of the total, whereas data editing and calculation tasks are 19% (22/115) and 25% (29/115) of the total, respectively.

Grading the participants' performance was done as follows. For tasks involving adding new data to the spreadsheet or performing calculations over spreadsheet data, whenever a participant executes a task as we asked him/her to, he/she is awarded 100% of the total score for that task; on the contrary, if the participant does not at all try to solve a particular task, he/she gets no credit for that. An intermediate situation occurs when participants try to solve a task, but fail to successfully conclude it in its entirety. In this case, the participant is awarded 50% of the score for that task. For tasks involving editing data, a value in the interval $0\% - 100\%$ is awarded according to the participants' success rate in such tasks. Table 2 shows the number of participants that worked on each spreadsheet and each model. Note that the distribution of models and spreadsheets by the participants is homogeneous.

Table 2: Participants per spreadsheet/model.

	original	*refactored*	*schema-aware*	Total
DISHES	12	13	12	37
PROJECTS	11	13	13	37
PROPERTIES	14	11	13	38
Total	37	37	38	

6.1.4. Hypotheses

In order to have an idea on how much relevant are the results obtained, we performed a statistical analysis. For that, we defined three kinds of hypotheses to test, one for each research question. Then, we tested each of this hypotheses with all combinations of models: *original* vs. *refactored* (represented by *orig. / ref.* in the tables), *original* vs. *schema-aware* (represented by *orig. / s.a.* in the tables), and *refactored* vs. *schema-aware* (represented by *ref. / s.a.* in the tables). The hypotheses are formulated as follows:

- The null hypothesis for **RQ1**, H_{RQ1_0}, is: the rate of correct tasks is equal between the two models tested. $H_{RQ1_0} : \mu_x = \mu_y$, where μ_x is the expected mean for the first model and μ_y is the expected mean for the second model.
  The alternate hypothesis, H_1, is that the rate of correct tasks is different between the two models tested, $H_{RQ1_1} : \mu_x \neq \mu_y$.
  This hypothesis is tested on each spreadsheet (*i.e.*, DISHES, PROJECTS, and PROPERTIES).
- The null hypothesis for **RQ2**, H_{RQ2_0}, is: the rate of correct tasks is equal between the two models tested. $H_{RQ2_0} : \mu_x = \mu_y$, where μ_x is the expected mean for the first model and μ_y is the expected mean for the second model.
  The alternate hypothesis, H_1, is that the rate of correct tasks is different between the two models tested, $H_{RQ2_1} : \mu_x \neq \mu_y$.
  This hypothesis is tested for each task (*i.e.*, *data insertion*, *data edition*, and *statistics*) on each spreadsheet (*i.e.*, DISHES, PROJECTS, and PROPERTIES).
- The null hypothesis for **RQ3**, H_{RQ3_0}, is: the time to perform the tasks is equal between the two models tested. $H_{RQ3_0} : \mu_x = \mu_y$, where μ_x is the expected mean time for the first model and μ_y is the expected mean time for the second model.
  The alternate hypothesis, H_3, is that the time to perform the tasks is different between the two models tested, $H_{RQ1_3} : \mu_x \neq \mu_y$.
  This hypothesis is tested on each spreadsheet (*i.e.*, DISHES, PROJECTS, and PROPERTIES).

6.2. Analyzing End-User Performance

We divide the presentation of our empiric results under two main axes: effectiveness and efficiency. In studying effectiveness we want to compare the three running models for the percentage of correct tasks that participants produced in each one. In studying efficiency we wish to compare the time that participants took to execute their assigned tasks in each of the different models. We start by effectiveness.

6.2.1. Effectiveness

Each participant was handed 3 different lists of tasks (insert, edit and query data) to perform on 3 different spreadsheets (DISHES, PROJECTS, PROPERTIES). Each spreadsheet, for the same participant, was constructed under a different model (*original*, *refactored schema-aware*).

For each spreadsheet, and for each model, we started by analyzing the average of the scores obtained by participants. Moreover, we tested the hypotheses H_E at the 0.05 significance level. The results of both the averages and p-values from the hypothesis testing are displayed in Figure 9.

We notice that no spreadsheet model is the best for all spreadsheets in terms of effectiveness. Indeed, we may even notice that spreadsheets in the traditional style, the *original* model, turned out to be the best for both the DISHES and PROPERTIES spreadsheets. The *schema-aware* model suited the best for the PROJECTS spreadsheet. However, no significant statistical differences were found, except between the effectiveness of the *original* model and the *schema-aware* one for the DISHES spreadsheet.

In the same line of reasoning, there is no worst model: *refactored* achieved the worst results for DISHES and PROJECTS; *schema-aware* got the lowest average scores for PROPERTIES. Nevertheless, these results seem to indicate that the models that we have developed are not effective in reducing the number of errors in spreadsheets, since one of them is always the model getting the lowest scores. This first intuition, however, deserves further investigation. For one, on the theoretical side, one may argue that *original* is, without a doubt, the *model* that end users are accustomed to. Recall that in the study, we opted to leave out participants with computer science backgrounds, who could be more sensible to the more complex models *refactored* and *schema-aware*, preferring to investigate such models on traditional users of spreadsheets. On the other hand, we remark that these more complex models were not introduced; a part of our study was also to learn whether or not they could live on their own.

	original	*refactored*	*schema-aware*
DISHES	86%	76%	78%
PROJECTS	73%	68%	78%
PROPERTIES	75%	64%	62%

p-values	orig. / ref.	orig. / s.a.	ref. / s.a.
DISHES	0.184	**0.0293**	0.838
PROJECTS	0.344	0.5622	0.111
PROPERTIES	0.287	0.2625	0.833

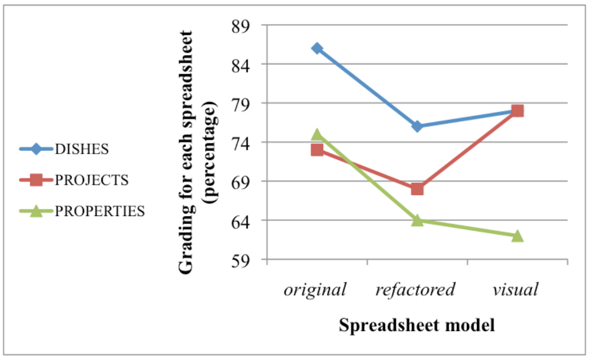


Figure 9: The p-values obtained from testing the hypotheses at the H_E 0.05 significance level, and the average of the effectiveness scores obtained by participants for each spreadsheet and for each model.

Our next step was to investigate whether the (apparent) poor results obtained by complex models are due to their own nature or if they result from participants not having understood them. So, we studied participations that did not achieve at least 50%, which are distributed by the spreadsheet models as follows: *original*, 0%, *refactored*, 25% and *schema-aware*, 21%.

While in *original* no participation was graded under 50%, this was not the case for *refactored* and *schema-aware*, which may have degraded their overall average results. For these participations, we analyzed the questionnaire that participants were asked to fill in after the session. The average classifications for the post session questionnaires, for participations in the study that were graded under 50% is 24% for *refactored* and 31% for *schema-aware*.

These results show that participants obtaining poor gradings on their effectiveness, also got poor gradings for their answers to the questions assessing how they understood the models they had worked with. In fact, such participants were not able to answer correctly (to at least) two thirds of the questions raised in the post session questionnaire. From such results we can read that 1/4 of participants was not able to understand the more complex models, which might have caused a degradation of the global effectiveness results for these models. This also suggests that if these models are to be used within an organization, it is necessary to take some time to introduce them to end users in order to achieve maximum effectiveness. Nevertheless, even without this introduction, the results show that the models are competitive in terms of effectiveness: at most they are 13% worse than the *original* model, and for one of the spreadsheets, the *schema-aware* model even got the best global effectiveness.

6.2.2. Effectiveness by Task Type

Next, we wanted to realize how effective models are to perform each of the different types of tasks that we have proposed to participants: *data insertion*, *data editing* and *statistics*.

i) *Data insertion:* The results presented in Figure 10 show, for each model, how effective participants were in adding new information to the spreadsheets they received. The results of hypothesis testing are also present, namely the p-values.

The *original* model revealed to be the most effective, for all three spreadsheets, being closely followed by *refactored* and *schema-aware* for DISHES, and by *schema-aware* for PROJECTS. The *refactored* model, for PROJECTS, and the models *refactored* and *schema-aware*, for PROPERTIES, proved not to be competitive for data insertion, in the context of the study. The differences that we obtained are not statistically significant, except when comparing *original* with *schema-aware* for DISHES. Again, we believe that this in part due to these models not having been introduced previously to the study: the insertion of new data is the task that is most likely to benefit from totally understanding of the running model, and also the one that can be otherwise most affected. This is confirmed by the effectiveness results observed for other task types, that we present next.

ii) *Data editing:* Now, we analyze the effectiveness of the models for editing spreadsheet data. The results presented in Figure 11 show that once a spreadsheet is populated, we can effectively use the models to edit its data.

	original	*refactored*	*schema-aware*
DISHES	91%	90%	81%
PROJECTS	76%	60%	75%
PROPERTIES	86%	67%	68%

p-values	orig. / ref.	orig. / s.a.	ref. / s.a.
DISHES	0.2938	**0.0091**	0.7501
PROJECTS	0.3401	0.5561	0.1075
PROPERTIES	0.0295	0.0378	0.9005

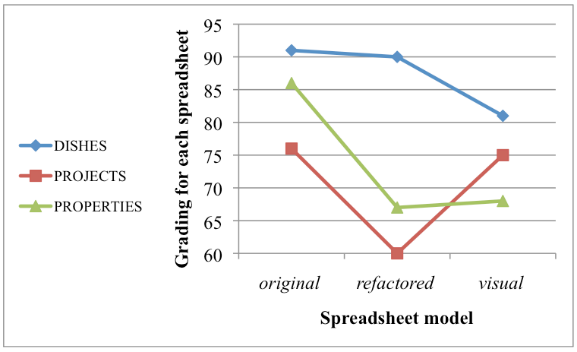


Figure 10: Effectiveness results for data insertion.

	original	*refactored*	*schema-aware*
DISHES	91%	82%	82%
PROJECTS	54%	62%	50%
PROPERTIES	65%	98%	48%

p-values	orig. / ref.	orig. / s.a.	ref. / s.a.
DISHES	0.202	0.455	0.622
PROJECTS	0.440	0.996	0.381
PROPERTIES	**0.008**	0.322	**0.002**

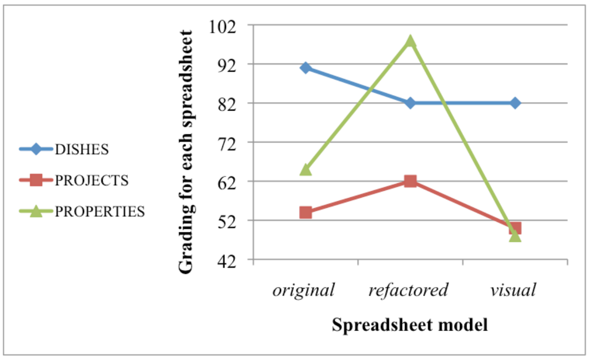


Figure 11: Effectiveness results for data edition.

This is the case of *refactored* for PROJECTS and specially for PROPERTIES where there is statistical significance in the differences comparing to the other two models. *Original* is the most effective in data editing for DISHES. *Schema-aware* is comparable to *refactored* for DISHES, but for all other spreadsheets, it always achieves the lowest scores among the three models.

iii) *Statistics:* Finally, we have measured the effectiveness of the models for performing calculations over spreadsheet data, obtaining the results shown in Figure 12.

	original	*refactored*	*schema-aware*
DISHES	52%	37%	57%
PROJECTS	19%	76%	13%
PROPERTIES	44%	57%	51%

p-values	orig. / ref.	orig. / s.a.	ref. / s.a.
DISHES	0.197	0.653	0.119
PROJECTS	0.289	0.372	0.878
PROPERTIES	0.375	0.676	0.687

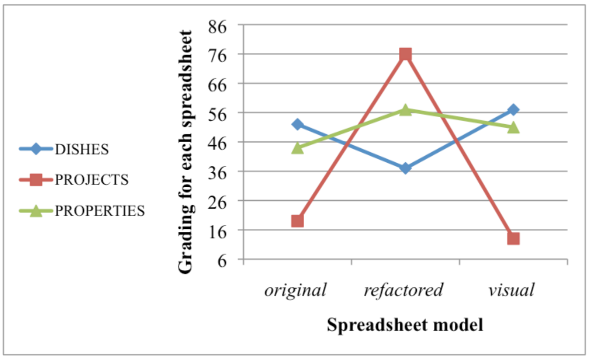


Figure 12: Effectiveness results for statistical calculations.

We can see that *schema-aware* obtained the best results for DISHES, and that *refactored* obtained the best results for both spreadsheets PROJECTS and PROPERTIES. We can also see that the *original* model was the worst model for

PROPERTIES, *refactored* was the worst model for DISHES, and *schema-aware* was the worst for PROJECTS, that is, each model got the worst result for one spreadsheet. However, no differences are statistically significant, as shown by the p-values in the figure.

Results from i), ii) and iii) confirm that the models are competitive. On the other hand, these results allow us to draw some new conclusions: if the models are going to be used within an organization, it may not always be necessary to introduce them prior to their use. Indeed, if an organization mostly edits spreadsheet data or computes new values from such data, and does not insert new data, then the models, and specially *refactored*, may deliver good results even without being explained. These results also show that it is inserting data that models need to be better understood by end users in order to increase effectiveness.

6.2.3. Efficiency

In this section, we analyze the efficiency results obtained in our study by the models that we have been considering in this paper.

We started by measuring, for each participant, and for each spreadsheet, the time elapsed from the moment participants started reading the list of tasks to undertake until the moment they completed the tasks proposed for that particular spreadsheet and moved on to a different spreadsheet or concluded the study. We are able to calculate these times by looking at the individual screen activity that was recorded during the study, for each participant: the participant stopping interacting with the computer signals the end of his/her work on a spreadsheet. The measured period therefore includes the time that participants took trying to understand the models they received each spreadsheet in. Figure 13 presents the average of the overall times, for each spreadsheet and for each model, along with the p-values resulting from the hypothesis tests.

	original	*refactored*	*schema-aware*
DISHES	35′	32′	28′
PROJECTS	39′	40′	41′
PROPERTIES	37′	36′	40′

p-values	orig. / ref.	orig. / s.a.	ref. / s.a.
DISHES	**0.022**	**0.062**	0.409
PROJECTS	0.639	0.278	0.542
PROPERTIES	0.622	0.630	0.444

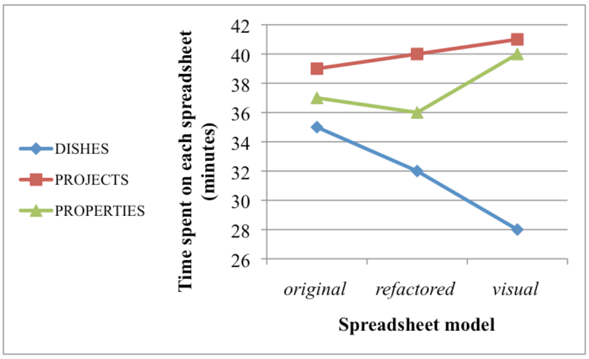


Figure 13: Global efficiency results.

We can see that *refactored* and *schema-aware* are competitive in terms of efficiency: participants performed fastest for DISHES in *schema-aware*, and fastest, by a marginal factor, for the PROPERTIES in *refactored*. *Original* got the best efficiency measurements for PROJECTS, also by a marginal factor. From the hypothesis testing, statistically significant differences were only obtained in the DISHES spreadsheet, when comparing the *original* model against the other two. Again, note that no introduction to these models preceded the study. Therefore, it is reasonable to assume that the results in Figure 13 include some time overhead. In an attempt to measure this overhead, which is a consequence of participants having to analyze a new model, we extracted some information out of the participants' screen activity. Indeed, we measured the time elapsed from the moment participants started reading, for each spreadsheet, the list of tasks to perform, until the moment they actually began editing the spreadsheet. We assume that this period corresponds exactly to the overhead of understanding each model (obviously increased by the time spent reading the list of tasks, which we are not able to isolate any further, but that should be constant for any spreadsheet model, since the task list does not change with the model). These results are presented in Table 3. We notice that there is a constant average overhead of 2 minutes for almost all models and spreadsheets, with the most significant exceptions occurring for *refactored*, for both DISHES and PROJECTS. In these cases, we can clearly notice an important time gap, which provides some evidence that *refactored* is most likely the hardest model to understand. This also comes in line with

	original	*refactored*	*schema-aware*
DISHES	$2'$	$6'$	$1'$
PROJECTS	$2'$	$4'$	$2'$
PROPERTIES	$2'$	$2'$	$2'$

Table 3: Average overhead results.

previous indications that the merits of models can be maximized if we explain them to end users. For the particular case of efficiency, this means that the results shown in Figure 13 could be further improved for the more complex models, and particularly for *refactored*.

6.3. Threats to Validuty

As suggested by Perry *et al.* [23], we discuss three types of influences that might limit the validity of our study.

Construct Validity: *Do the variables and hypotheses of our study accurately model the research questions?*

i) *Measuring the time overhead:* when studying efficiency, we measured the overhead of understanding each model as the period of time that participants stopped interacting with a spreadsheet and started editing the next one. In this period, it might have been the case that participants, instead of being focused on understanding the new model, took the time to do something else, like resting. This could affect our conclusions in terms of efficiency. However, during the study, participants where supervised by two authors, who observed that this was not the case. Even if we were not able to spot a small number of such occurrences, the differences in the results should be minimal and so they should not affect our conclusions.

ii) *Original model*: In our study, we have used three spreadsheets that we have assumed to be in the *original* model. What we are saying is that these three spreadsheets are representative of the spreadsheets normally defined by end users. Although this set of spreadsheets may be too large to be represented by (any) three spreadsheets, we have taken DISHES, PROJECTS and PROPERTIES directly, or with small changes, from other works on general purpose spreadsheets [25, 11, 24].

Internal Validity: *Can changes in the dependent variables be safely attributed to changes in the independent variable?*

i) *Accuracy of the analysis*: Some of the inferences we make in this paper deserve further analysis. To some extent, we assume that our models could achieve better results if a tutorial has been given to the participants. In fact, we have no proof of this, but the evidences from the study seem to strongly indicate this fact. A new study is required to prove this, though.

ii) *Accuracy of measurements*: Each task proposed to participants was individually graded. For most of the cases, this was done automatically using OpenOffice scripts. These scripts and their results were tested and checked. The cases for which an automatic grading was not possible were carefully graded by hand. All grades were validated by two authors and were randomly re-checked. Since we have more than 1400 grades, it is virtually impossible to guarantee full accuracy. This could affect the results observed for dependent variables (efficiency and effectiveness) without really the independent variables (the models considered) having changed. Nevertheless, if imprecisions exist in the grades, they should be equally distributed by the 3 models and thus they should not affect the overall results.

The measurement of times that lead to the results presented earlier was achieved by visualizing the screen casts made during the study. Being a manual and repetitive task, it is subject to imprecisions. Also, not being able to visualize the actual participants' behaviour now may lead to imprecise measurements. We are confident that, even if there are imprecisions, such imprecisions should be distributed evenly by all measurements and thus do not influence the efficiency results or the conclusions that we draw based on them.

External Validity: *Can the study results be generalized to settings outside the study?*

$i)$ *Generalization*: In this study we used three different spreadsheets from different domains. We believe that the results can be generalized to other spreadsheets, although probably not to all. The models we developed are not restricted to any particular spreadsheet, and thus, the results should be the same if the study was run with a different set of spreadsheets.

$ii)$ *Industrial usage*: Participants were asked to simulate industrial activity: they received some data on paper that they had to register in a spreadsheet. Although we have tried to create a realistic environment for the study, it is likely that people would respond differently in an industrial context. Also, participants were University students whose technical abilities and experience surely differ from other spreadsheet users. Nevertheless, we believe that this affects no spreadsheet/model in particular. Possible impacts would affect all spreadsheets/models in the same way and thus the overall results apply. We believe that if the study was conducted in an industrial environment, the conclusions should be similar.

6.4. Answering the Research Questions

In this section we discuss the results obtained in our study and answer the proposed research questions.

According to [23], three topics deserve further analysis. The first is *accuracy of interpretation*: this study was prepared carefully and a significantly large number of end users participated in it. Our goal here was to guarantee that the results are not unknowingly influenced. For this, it also contributes the fact that we make all the elements of this study available, both in this paper and online.

The second topic is *relevance*: Model-Driven Engineering (MDE) is one of the most significant research areas in software engineering. We adapted some techniques from this field to spreadsheets and showed that they can bring benefit not only for professional users but also for end users.

The last topic is *impact*: our first results show that MDE can bring benefits for spreadsheet end users. This is a promising research direction, that we believe can be further explored, particularly in contexts similar to the one of this paper.

We now seek to answer the research questions that we presented in the introduction of this paper, which correspond exactly to the questions our study was designed to answer.

RQ1: The data collected shows that for effectiveness our techniques helped users for one of the spreadsheets. Although this is not a great result, it shows that it is possible to help end users. Nevertheless, more work needs to be done to generalize this result to other spreadsheets. Moreover, our results also indicate that deeper insight on the spreadsheet models is required to maximize effectiveness. Indeed, we believe that the effectiveness results for *refactored* and *schema-aware* could have been significantly better if these models had been preliminary presented to the participants of our study.

RQ2: For inserting tasks, the spreadsheets our techniques produced could only be comparable to the *original* one in two cases. In the third case, although not very significantly, the *original* got better results. For editing, in two spreadsheets users were more effective using our techniques and in the third one, the difference was not very significant. For querying, the spreadsheets we produce got the better results. Furthermore, the results seem to indicate that the inserting data task is the one that benefits the most from better understanding the models. Indeed, in the last task performed by the users, that is, querying the data, they got very good results using our spreadsheets, which seems to indicate that the more they know and work with them, the better they get. Thus, we can answer to this questions by saying: first, when inserting data no help was actually given by our spreadsheets, and indeed for one spreadsheet we got the worst results, and for the other two we were only 1% worst; second, when editing data, the results obtained using our spreadsheets were already better for two spreadsheets, thus helping end users; third, users were greatly helped by our techniques when querying data since for the three spreadsheets our techniques achieved better results than the *original* one.

RQ3: We observed that, frequently, the more elaborate spreadsheet models allowed users to perform faster. Nevertheless, we were not fully able of isolating the time that participants took trying to understand the models they were given. So, we believe that the observed efficiency results could also be better for *refactored* and *schema-aware* if they had been introduced.

With this study we have shown that there is potential in MDE techniques for helping spreadsheet end users. The study of these techniques for professional users of spreadsheets seems a promising research topic. Moreover, the use of MDE techniques in other non-professional softwares should also be investigated.

7. Related Work

Our work is strongly related to a series of techniques by Abraham *et al.* Firstly, they designed and implemented an algorithm that uses the labels within a spreadsheet for *unit checking* [26, 27]. By typing the cells in a spreadsheet with unit information and tracking them through references and formulas, various types of user errors can be caught. We have adopted the view of Abraham *et al.* of a spreadsheet as a collection of tables and we have reused their algorithm for identifying the spatial boundaries of these tables. Rather than exploiting the labels in the spreadsheet to reconstruct implicit user intentions, we exploit redundancies in data elements. Consequently, the errors caught by our approach are of a different kind. Secondly, Abraham *et al.* developed a type system and corresponding inference algorithm that assigns types to values, operations, cells, formulas, and entire spreadsheets [28]. The type system can be used to catch errors in spreadsheets or to infer spreadsheet models that can help to prevent future errors. We have used such spreadsheet models, namely the ClassSheet models [29], to realize model-driven software evolution in the context of spreadsheets [22, 30, 31, 32, 33, 34, 35].

Cheng and Rival propose an abstract domain to infer types of parts (for instance, columns) of a spreadsheet [36]. In fact they consider the code to populate a spreadsheet, and not the spreadsheet data as displayed in a spreadsheet environment as we do. The language they consider is similar to Excel VBA and OpenOffice Basic, but accounts only for a subset of such languages. The evaluation they present is performance-oriented, which is different from ours.

In previous work we presented techniques and tools to transform spreadsheets into relational databases and back [7]. We used the FUN algorithm to construct a relational model, but rather than generating edit assistance, the recovered information was used to perform spreadsheet *refactoring*. The algorithm for extracting and filtering spreadsheets presented in the current paper is an improvement over the algorithm that we used previously.

We provided a short user-centered overview of the idea of generating edit assistance for spreadsheets via extraction of functional dependencies in a previous short paper [37]. In the current paper, we have provided the technical details of the solution, including the improved algorithm for extraction and filtering functional dependencies. Also, we have provided the first preliminary evaluation of the approach by application to a sample of spreadsheets from the EUSES corpus.

8. Conclusions

Contributions. We have demonstrated how implicit structural properties of spreadsheet data can be exploited to offer edit assistance to spreadsheet users. To discover these properties, we have made use of our improved approach for mining functional dependencies from spreadsheets and subsequent synthesis of a relational database. On this basis, we have made the following contributions:

- Derivation of formulas and visual elements that capture the knowledge encoded in the reconstructed relational database schema.
- Embedding of these formulas and visual elements into the original spreadsheet in the form of features for auto-completion, guarded deletion, and controlled insertion.
- Generation of a new refactored spreadsheet with no data redundancy that guides the user inserting and editing correct data.
- Integration of the algorithms for reconstruction of a schema, for derivation of corresponding formulas and visual elements, and for their embedding into a *add-in* for spreadsheet environments.

- We have shown that MDE techniques can be adapted for end-users' software. Unfortunately, the results achieve can still be improved as users could not gain efficiency and effectiveness has it would be desirable. Although our approach is valid from a computer science perspective, it is not straightforward to make it available for end users, as this would probably require them to have more knowledge about advanced data manipulation.
- Finally, we have proposed a methodology that can be reused in studies similar to the one we have conducted.

Future Work. There are several extensions of our work that we would like to explore. The algorithms running in the background need to recalculate the relational schema and the ensuing formulas and visual elements every time new data is inserted. For larger spreadsheets, this recalculation may incur waiting times for the user. Several optimizations of our algorithms can be attempted to eliminate such waiting times, for example, by using of incremental evaluation. Our approach could be integrated with similar, complementary approaches to cover a wider range of possible user errors. In particular, the work of Abraham *et al.* [28, 38] for preventing range, reference, and type errors could be combined with our work for preventing data loss and inconsistency. Moreover, the results in terms of effectiveness and efficiency can be improved. We believe that with some training these results can be greatly improved. We plan to create a new experiment where training is given to participants. If good results come from this new study, then we can conclude that MDE techniques can help end users.

Acknowledgment

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project `FCOMP-01-0124-FEDER-010048`. The first author is supported by the FCT grant `SFRH/BPD/73358/2010`.

References

[1] J. Saraiva, Design, Implementation and Animation of Spreadsheets in the Lrc System, in: M. Erwig (Ed.), International Workshop on Foundations of Spreadsheet, ENTCS, 2004.

[2] M. Kuiper, J. Saraiva, Lrc - A Generator for Incremental Language-Oriented Tools, in: K. Koskimies (Ed.), 7th International Conference on Compiler Construction, Vol. 1383 of LNCS, Springer-Verlag, 1998, pp. 298–301.

[3] T. Reps, T. Teitelbaum, The synthesizer generator, SIGSOFT Software Engineering Notes 9 (3) (1984) 42–48.

[4] M. van den Brand, P. Klint, P. Olivier, Compilation and Memory Management for ASF+SDF, in: Stefan Jähnichen (Ed.), 8th International Conference on Compiler Construction, Vol. 1575 of LNCS, 1999, pp. 198–213.

[5] J. Saraiva, S. D. Swierstra, Generating spreadsheet-like tools from strong attribute grammars, in: F. Pfenning, Y. Smaragdakis (Eds.), Generative Programming and Component Engineering, Second International Conference, GPCE 2003, Erfurt, Germany, September 22-25, 2003, Proceedings, Vol. 2830 of LNCS, Springer, 2003, pp. 307–323.

[6] S. Holzner, Eclipse, O'Reilly, 2004.

[7] J. Cunha, J. Saraiva, J. Visser, From spreadsheets to relational databases and back, in: Proceedings of the 2009 ACM SIGPLAN Workshop on Partial evaluation and Program Manipulation, PEPM '09, ACM, New York, NY, USA, 2009, pp. 179–188.

[8] J. Cunha, J. Saraiva, J. Visser, Model-based programming environments for spreadsheets, in: F. de Carvalho Junior, L. Barbosa (Eds.), Brazilian Symposium on Programming Languages, Vol. 7554 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2012, pp. 117–133.

[9] E. F. Codd, A relational model of data for large shared data banks., Commun. ACM 13 (6) (1970) 377–387.

[10] M. F. II, G. Rothermel, The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms, in: Proceedings of the 1st Workshop on End-User Software Engineering, 2005, pp. 47–51.

[11] T. Connolly, C. Begg, Database Systems, A Practical Approach to Design, Implementation, and Management, Addison-Wesley, 3 edition, 2002.

[12] J. D. Ullman, J. Widom, A First Course in Database Systems, Prentice Hall, 1997.

[13] C. J. Date, An Introduction to Database Systems, Addison-Wesley, 1995.

[14] C. Beeri, R. Fagin, J. Howard, A complete axiomatization for functional and multivalued dependencies in database relations., in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1977, pp. 47–61.

[15] D. Maier, The Theory of Relational Databases, Computer Science Press, 1983.

[16] J. Cunha, Model-based spreadsheet engineering, Ph.D. thesis, University of Minho (March 2011).

[17] S. Peyton Jones, Haskell 98: Language and libraries, Journal of Functional Programming 13 (1) (2003) 1–255.

[18] F. Hermans, M. Pinzger, A. v. Deursen, Detecting and visualizing inter-worksheet smells in spreadsheets, in: Proceedings of the 2012 International Conference on Software Engineering, ICSE 2012, IEEE Press, Piscataway, NJ, USA, 2012, pp. 441–451.

[19] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, E. Getzner, On the empirical evaluation of fault localization techniques for spreadsheets, in: Proceedings of the 16th international conference on Fundamental Approaches to Software Engineering, FASE'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 68–82.

[20] C. Chambers, M. Erwig, Reasoning about spreadsheets with labels and dimensions, Journal of Visual Languages and Computing 21 (5) (2010) 249–262.

[21] R. Abraham, M. Erwig, UCheck: A spreadsheet type checker for end users., J. Vis. Lang. Comput. 18 (1) (2007) 71–95.

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

[23] D. E. Perry, A. A. Porter, L. G. Votta, Empirical studies of software engineering: a roadmap, in: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, ACM, New York, NY, USA, 2000, pp. 345–355.

[24] S. G. Powell, K. R. Baker, The Art of Modeling with Spreadsheets, John Wiley & Sons, Inc., New York, NY, USA, 2003.

[25] R. Alhajj, Extracting the extended entity-relationship model from a legacy relational database, Information Systems 28 (6) (2003) 597 – 618.

[26] M. Erwig, M. Burnett, Adding apples and oranges, 4th International Symposium on Practical Aspects of Declarative Languages (2002) 173–191.

[27] R. Abraham, M. Erwig, Header and unit inference for spreadsheets through spatial analyses, Visual Languages and Human Centric Computing, 2004 IEEE Symposium on (2004) 165–172.

[28] R. Abraham, M. Erwig, Type inference for spreadsheets, in: A. Bossi, M. J. Maher (Eds.), Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, ACM, 2006, pp. 73–84.

[29] G. Engels, 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, ACM, 2005, pp. 124–133.

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

[31] J. Cunha, J. P. Fernandes, J. Mendes, J. Saraiva, Towards an evaluation of bidirectional model-driven spreadsheets, in: User evaluation for Software Engineering Researchers, USER' 12, ACM Digital Library, 2012, pp. 25–28.

[32] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, J. Saraiva, Bidirectional transformation of model-driven spreadsheets, in: Z. Hu, J. de Lara (Eds.), Theory and Practice of Model Transformations, Vol. 7307 of Lecture Notes in Computer Science, Springer, 2012, pp. 105–120.

[33] J. Cunha, J. P. Fernandes, J. Mendes, J. Saraiva, MDSheet: A Framework for Model-driven Spreadsheet Engineering, in: Proceedings of the 34rd International Conference on Software Engineering, ICSE'12, ACM, 2012, pp. 1412–1415.

[34] J. Cunha, J. Visser, T. Alves, 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.

[35] J. Cunha, J. P. Fernandes, J. Mendes, J. Saraiva, Extension and implementation of ClassSheet models, in: Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing, VLHCC '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 19–22.

[36] T. Cheng, X. Rival, An abstract domain to infer types over zones in spreadsheets, in: A. Miné, D. Schmidt (Eds.), Static Analysis, Vol. 7460 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 94–110.

[37] J. Cunha, J. Saraiva, J. Visser, Discovery-based edit assistance for spreadsheets, in: Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), VLHCC '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 233–237.

[38] R. Abraham, M. Erwig, Inferring templates from spreadsheets, in: Proceedings of the 28th International Conference on Software Engineering, ICSE '06, ACM, New York, NY, USA, 2006, pp. 182–191.