Model Inference for Spreadsheets

Jácome Cunha · Martin Erwig ·
Jorge Mendes · João Saraiva

Abstract Many errors in spreadsheet formulas can be avoided if spreadsheets are built automatically from higher-level models that can encode and enforce consistency constraints in the generated spreadsheets. Employing this strategy for legacy spreadsheets is difficult, because the model has to be reverse engineered from an existing spreadsheet and existing data must be transferred into the new model-generated spreadsheet.

We have developed and implemented a technique that automatically infers relational schemas from spreadsheets. This technique uses particularities from the spreadsheet realm to create better schemas. We have evaluated this technique in two ways: First, we have demonstrated its applicability by using it on a set of real-world spreadsheets. Second, we have run an empirical study with users. The study has shown that the results produced by our technique are comparable to the ones developed by experts starting from the same (legacy) spreadsheet data.

Although relational schemas are very useful to model data, they do not fit well spreadsheets as they do not allow to express layout. Thus, we have also introduced a mapping between relational schemas and ClassSheets. A ClassSheet controls further changes to the spreadsheet and safeguards it against a large class of formula errors. The developed tool is a contribution to spreadsheet (reverse) engineering, because it fills an important gap and allows a promising design method (ClassSheets) to be applied to a huge collection of legacy spreadsheets with minimal effort.

Keywords Spreadsheets · ClassSheets · Relational Model · Automatic Model Inference · Empirical Validation

1 Introduction

Spreadsheets are one of the most popular programming systems, especially concerning business applications. Every year, hundreds of millions of spreadsheets are created by business users, and numerous studies show that this high rate of production is accompanied by an alarming high rate

Jácome Cunha
HASLab/INESC Tec & Universidade Nova de Lisboa, Portugal
E-mail: jacome@fct.unl.pt

Martin Erwig
School of EECS, Oregon State University, USA
E-mail: erwig@eecs.oregonstate.edu

Jorge Mendes
HASLab/INESC Tec & Universidade do Minho, Portugal
E-mail: jorgemendes@di.uminho.pt

João Saraiva
HASLab/INESC Tec & Universidade do Minho, Portugal
E-mail: jas@di.uminho.pt

of errors (EuSpRIG, 2010; Panko, 2000; Powell and Baker, 2003). Some studies report that up to 90% of real-world spreadsheets contain errors (Rajalingham et al, 2001).

Spreadsheet systems offer users a high level of flexibility, making it easier for people to get started working with spreadsheets. The downside is that this freedom also offers ample opportunity to create erroneous spreadsheets. Errors during creation of a spreadsheet are made as well as when modified by other users. The problem gets exacerbated when the people who use or modify the spreadsheet do not fully understand its functionality. This situation arises because spreadsheet systems do not offer any higher-level abstractions like modern programming languages do. Moreover, data and computation are not separated in spreadsheets, and the immediate visual feedback mechanism makes traditional coding and program compilation/execution steps indistinguishable from each other. These factors make widespread reuse of spreadsheets difficult and prone to errors.

In recent years the spreadsheet research community has recognized the need to support *end-user, model-driven software development*, and to provide spreadsheet developers and end users with methodologies, techniques and the necessary tool support to improve their productivity. In fact, several techniques have been proposed to allow end users to safely edit spreadsheets, like, for example, the use of spreadsheet templates (Abraham et al, 2005), ClassSheets (Engels and Erwig, 2005), and the inclusion of visual objects to provide editing assistance in spreadsheets (Cunha et al, 2009a). All these approaches aim at a form of model-driven software development: First, a spreadsheet business model is defined, from which then a customized spreadsheet application is generated that guarantees the consistency of the spreadsheet with the underlying model.

Despite of its huge benefits, model-driven software development is sometimes difficult to realize in practice. For example, in the context of spreadsheets, the use of model-driven software development requires that the developer is familiar both with the spreadsheet domain (business logic) and with model-driven software development. As some studies suggest, defining the business model of a spreadsheet can be a complex task for end users (Abraham and Erwig, 2006). As a result, end users are unable (or reluctant) to follow this spreadsheet development discipline. Things get even more complex when end users need to modify a large (legacy) spreadsheet developed by others and whose functionality they do not understand.

In this paper we propose reverse engineering techniques to derive relational models from existing spreadsheets. We use data mining techniques to reason about spreadsheet data and to infer functional dependencies among columns. These functional dependencies are the building blocks to infer the spreadsheet business model as a database schema. We present this work in Section 3. Section 4 presents an empirical validation of our approach when compared to professional programmers. Indeed the models our tool can automatically infer are comparable to the ones produced by professionals. Furthermore, an evaluation of the applicability of these techniques to a large set of spreadsheets can be found in Section 5.

Relational schemas however are not quite suitable for integration with spreadsheets as they do not allow to express layout constraints which are quite relevant for spreadsheets. Thus we have also developed a mapping from relational models to ClassSheets, which we present in Section 6. Finally, we have integrated these techniques under the MDSheet framework as explained in Section 7.

The remaining of this paper is organized as follows. In Section 2 we present a motivating example. Related work is discussed in Section 8, and Section 9 concludes the paper.

2 An Example

Consider the example spreadsheet shown in Figure 1, taken from (Powell and Baker, 2003). This spreadsheet represents a sales system for dishwasher detergents. The abbreviated labels have the following meanings: Label *com_code* stands for commercial code, *upc* for universal product code, *case* for number of cases, *move* for sales quantity, *qty* is a transaction quantity (which in this dataset is always set to 1), *ok* is a confirmation code, and *profit* is the profit and is calculated using the formula =I2*K2*0.2 (for the first row).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	com_code	upc	description	size	case	nitem	store	week	move	qty	price	onsale	profit	ok
2	653	1111140009	DOVE DISH LIQUID	42 OZ	9	2851281	100	383	16	1	2.19		7.01	1
3	653	1111140009	DOVE DISH LIQUID	42 OZ	9	2851281	100	384	7	1	2.19		3.07	1
4	653	1111140009	DOVE DISH LIQUID	42 OZ	9	2851281	100	385	15	1	2.19		6.57	1
5	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6	654	1111165003	SUNLIGHT AUTO GEL	88 OZ	6	2857061	100	390	6	1	3.75		4.5	1
7	654	1111165003	SUNLIGHT AUTO GEL	88 OZ	6	2857061	100	391	11	1	3.39	S	7.46	1
8	654	1111165003	SUNLIGHT AUTO GEL	88 OZ	6	2857061	100	392	8	1	3.75		6	1
9	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Fig. 1 A spreadsheet representing a sales system for dishwasher detergents.

The business logic that underlies this spreadsheet is not immediately clear, and is quite difficult to infer for a non-expert in this domain. In this section we will informally describe a strategy to infer such a business logic from the spreadsheet data.

Objects that are contained in such a spreadsheet and the relationships between them are reflected by the presence of *functional dependencies* between spreadsheet columns. A functional dependency between a column C and another column C' means that the values in column C determine the values in column C', that is, there are no two rows in the spreadsheet that have the same value in column C but differ in their values in column C'. This idea can be extended to multiple columns, that is, when any two rows that agree in the values of columns $C_1, \ldots, C_n$ also agree in their value in columns $C'_1, \ldots, C'_m$, then $C'_1, \ldots, C'_m$ are said to be *functionally dependent* on $C_1, \ldots, C_n$.

It is possible to construct a relational model from a set of observed functional dependencies. Such a model consists of a set of relation schemas (each given by a set of column names) and expresses the basic business model present in the spreadsheet. Each relation schema of such a model basically results from grouping functional dependencies together. For example, for the spreadsheet in Figure 1 we could infer the following relational model (underlined column names indicate those columns on which the other columns are functionally dependent, and columns preceded by # indicate foreign keys).

$ConfirmationCode$ $(\underline{ok})$
$Quantity$ $(\underline{qty})$
$OnSale$ $(\underline{store}, \underline{week}, onsale)$
$Profit$ $(\underline{move}, \underline{price}, profit)$
$Detergent$ $(\underline{com\_code}, upc, description, size, case, nitem)$
$Sale$ $(\#com\_code, \#move, \#price, \#store, \#week, \#qty, \#ok)$

The model has several relations: *ConfirmationCode* and *Quantity* contain only one value, the number 1. The entity *OnSale* registers in each week if a store is on sales. *Profit*, which comes directly from the formula in the column with the same name, has information about the profits based on the number of movements and price. *Detergent* contains all the information about the existing detergents. Finally, *Sale* stores the information on sales. Note that this model was automatically computed by our algorithm.

Although several models could be created to represent this system, we show in Section 4 that the models our tool automatically generates are comparable in quality to the ones designed by database experts. Moreover, we will see in Section 5 that our algorithm is applicable to a wide range of spreadsheets.

Although a relational model is very expressive, it is not quite suitable for spreadsheets since spreadsheets need to have a layout specification. In contrast, the ClassSheet modeling framework offers high-level, object-oriented formal models to specify spreadsheets and thus presents a promising alternative (Engels and Erwig, 2005). ClassSheets allow users to express business object structures within a spreadsheet using concepts from the Unified Modeling Language (UML). A spreadsheet application consistent with the model can be automatically generated, and thus a large variety of errors can be prevented.

We therefore employ ClassSheets as the underlying modeling approach for spreadsheets and transform the inferred relational model into a ClassSheet model. Figure 2 presents a ClassSheet modeling the running example.

	A	B	C	D	E	F	G	H
1	Sale		DetergentKey					...
2			Com_code	Ok	Qty	Move	Price	
3			com_code=Detergent.com_code	ok=ConfirmationCode.ok	qty=Quantity.qty	move=Profit.move	price=Profit.Price	...
4	OnSaleKey							...
5	store	week	related					...
6	store=OnSale.store	week=OnSale.week	related=0					...
7	⋮	⋮	⋮	⋮	⋮		⋮	...
8								
9	Detergent							
10	Com_code	Upc	Description	Size	Case	Nitem		
11	com_code=0	upc=0	description=""	size=""	case=0	nitem=0		
12	⋮	⋮	⋮	⋮	⋮			
13								
14	OnSale				ConfirmationCode			
15	Store	Week	Onsale		Ok			
16	store=0	week=0	onsale=""		ok=0			
17	⋮	⋮	⋮		⋮			
18								
19	Profit				Quantity			
20	Move	Price	Profit		Qty			
21	move=0	price=0	profit=move*price*0.2		qty=0			
22	⋮	⋮	⋮		⋮			

Fig. 2 A ClassSheet modeling a sales system for dishwasher detergents.

The ClassSheet shown in Figure 2 consists of several *classes*. The simplest ones are the **ConfirmationCode** (`E14` to `E17`) and **Quantity** (`E19` to `E22`). Both classes have the corresponding name in the first row, a label for its unique column, and an attribute with the corresponding default value (0 in both cases). The last cell in grey and labeled with the ellipsis means these classes are *expandable vertically*, that is, they can have as many entries as necessary. These entries are added in rows.

The **Detergent** and **OnSale** classes have a similar structure, although these classes have more columns. Each column has a label, an attribute, and a corresponding default value. For text attributes the empty string is usually used as default value.

Profit has also a similar structure as the previous classes, but the attribute **profit** has not a default value but a formula to calculate its value. In this case the formula is the multiplication of the values in column **Move** by the ones in **Price** and by a constant (0.2).

Finally, the first seven rows represent the relationship **Sale**. Since **Sale** is a relationship between other relations, it is represented as a *cell class* (cells `C4` to `G6`), and the two related classes are **Detergent** and **OnSale**. In fact, since both relations **Detergent** and **OnSale** are already represented in the spreadsheet, only their keys are used. The remaining attributes in columns `D` to `G` are the other attributes of the relationship. The cell with the **related** attribute flags if a detergent and remaining attributes in the relationship are related to a particular pair of store/week. This is similar to have in a database a table where the primary key is the combination of the keys of the tables it combines. We will further discuss the spatial arrangement and placement of these classes in the spreadsheet in Section 6. The ellipsis filling row 7 indicates this class is expandable vertically. The ellipsis labeling column `H` indicate this class is *expandable horizontally*. Tables **Detergent** and **OnSale** were factored out in this way to avoid update anomalies, which occur when data is changed in one place and not changed in other places in the exactly same way (Ullman and Widom, 1997).

In Figure 3 we present a new spreadsheet generated from the ClassSheet modeling our running example. This spreadsheet has two entries for sales: One for store 100 and week 383 and another for the same store and week 384. There are also only two detergents represented. The profit is only shown for two movement/prices. As in the original spreadsheets, there is only one value for quantity and confirmation code.

ClassSheets carry information rich enough that allows the automatic generation of UML class diagrams from them. Figure 4 shows the UML class diagram that is derived from the ClassSheet from Figure 2. For more details about this transformation the reader is referred to (Cunha et al, 2012d).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Sale		DetergentKey					DetergentKey					
2			Com_code	Ok	Qty	Move	Price	Com_code	Ok	Qty	Move	Price	
3			653	1	1	16	2.19	653	1	1	7	2.19	
4	OnSaleKey												Add
5	store	week	related					related					DetergentKey
6	100	383		1					0				
7	100	384		0					1				
8					Add OnSaleKey								
9													
10	Detergent												
11	Com_code	Upc	Description	Size	Case		Nitem						
12	653	1111140009	DOVE DISH LIQUID	42 OZ	9		2851281						
13	654	1111165003	SUNLIGHT AUTO GEL	88 OZ	6		2857061						
14			Add Detergent										
15													
16	OnSale				ConfirmationCode								
17	Store	Week	Onsale		Ok								
18	100	383	""		1								
19	100	384	""		Add ConfirmationCode								
20			Add OnSale										
21													
22	Profit				Quantity								
23	Move	Price	Profit		Qty								
24	16	2.19	7.01		1								
25	7	2.19	3.07		Add Quantity								
26			Add Profit										

Fig. 3 A spreadsheet generated from the ClassSheet modeling the running example.
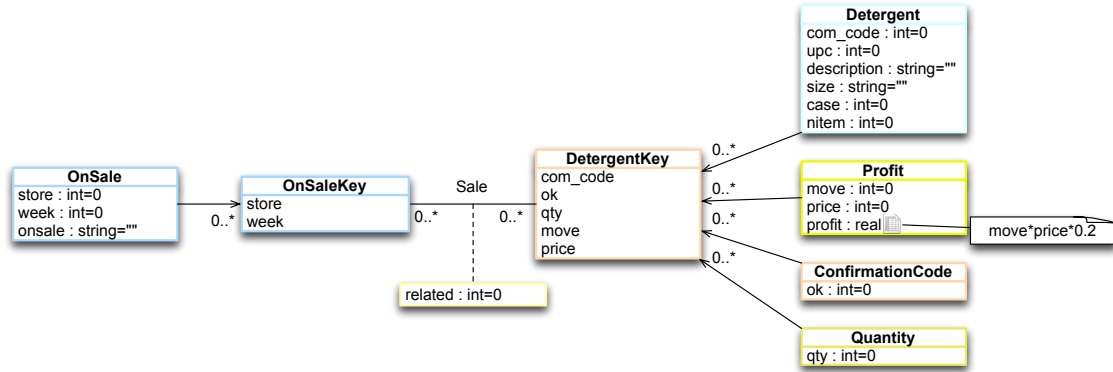

Fig. 4 A UML class diagram representing the business logic for the running example.

In this section, we have described our approach to derive relational schemas for spreadsheets through an example. We have also shown that from a relational model we can further construct a more suitable specification for spreadsheets, namely using ClassSheets. In fact, the database schema, the ClassSheet, and the UML class diagram were automatically produced by the tool we have implemented based on our approach. In the next sections we describe the steps and algorithms to infer relational models in detail. We also validate this in two different ways: First regarding its applicability, and second regarding its results. Finally, we will show how to map such models to ClassSheets and how they can be used to improve spreadsheets. The architecture of our approach is sketched in Figure 5.

3 Inferring Relational Schemas from Spreadsheets

In this section we explain in detail the steps to automatically extract an entity-relationship model from a spreadsheet. Essentially, our method involves the following steps:

1. Detect all functional dependencies and identify model-relevant functional dependencies.
2. Determine relational schemas with candidate, foreign, and primary keys.
3. Generate and refactor a relational graph representing the relational schema.
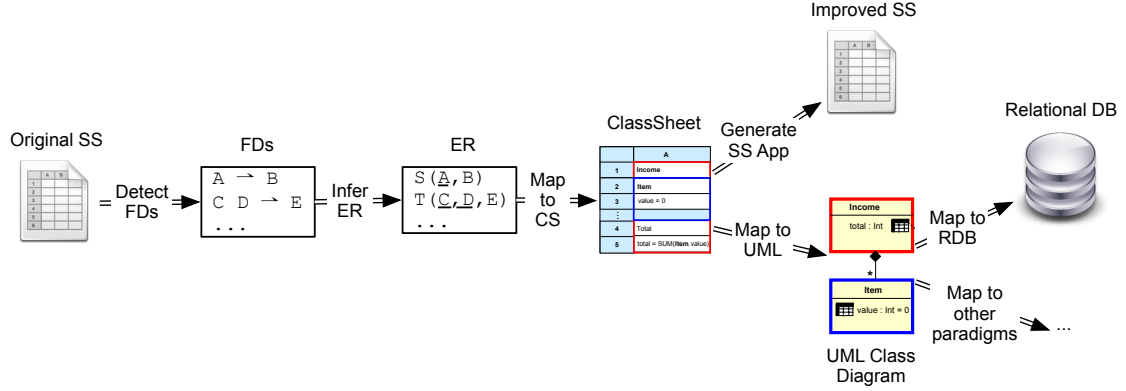
Fig. 5 An overview of the process to infer a ClassSheet and a UML class diagram.

In the following sub-sections we will explain the steps 1 and 3. Step 2 is a standard inference procedure borrowed from relational database theory (Alhajj, 2003).

3.1 Detecting Functional Dependencies

Knowledge about the functional dependencies in a spreadsheet provides the basis for identifying tables and their relationships in the data, which form the basis for relational models. The more accurate we can make this inference step, the better the inferred models will reflect the actual business logic. In this subsection we will therefore first repeat some definitions regarding relational models and functional dependencies and then describe a set of heuristics that are key to inferring valid dependencies.

A (relational) schema is a finite set of attributes. In the context of spreadsheets an attribute corresponds to the label of a column, such as *upc* and *week*. A (relational) table over a schema is given by a finite set of tuples, where tuples correspond to (contiguous parts of) spreadsheet rows (or columns). Our example includes the tables *Detergent* and *Sale*. Each tuple is uniquely identified by a set of attributes called *primary key*. For example, in the *Detergent* table this set is {*com_code*}. If a table has more than one set of attributes that can serve as primary key, called *candidate keys*, one of those has to be chosen as the primary key. A *foreign key* is a set of attributes within one table that matches the primary key of some other table.

A *functional dependency* between two sets of attributes A and B, written $A \rightharpoonup B$, holds in a table if for any two tuples t and t' in that table $t[A] = t'[A] \implies t[B] = t'[B]$ where $t[A]$ yields the (sub)tuple of values for the attributes in A. In other words, if the tuples agree in the values for attribute set A, they agree in the values for attribute set B. The attribute set A is also called *antecedent*, and the attribute set B *consequent*. For instance, in our running example the dependency *com_code* $\rightharpoonup$ *description* exists, meaning that the values in column *com_code* determine the values in column *description*.

Our goal is to use the data in a spreadsheet to identify functional dependencies. Although we use all the data available in the spreadsheet, we consider a particular instance of the spreadsheet domain only. However, there may exist counter examples to the dependencies found, but these just happen not to be included in the spreadsheet. Thus, the dependencies we discover are always an approximation. On the other hand, depending on the data, it can happen that many "accidental" functional dependencies are detected, that is, functional dependencies that do not reflect the underlying model. For instance, in our example we can identify the following dependency that just happens to be fulfilled for this particular data set, but that does certainly *not* reflect a constraint that should hold in general: *size* $\rightharpoonup$ *com_code case qty ok*. In fact, the data contained in the spreadsheet example supports around 30 functional dependencies. Here are a few more.

$$store \qquad \rightharpoonup qty\ ok$$
$$com\_code \quad \rightharpoonup qty\ ok$$
$$nitem \qquad \rightharpoonup com\_code\ size\ case\ qty\ ok$$
$$upc \qquad \rightharpoonup com\_code\ description\ size\ case\ nitem\ qty\ ok$$
$$move\ price \rightharpoonup profit$$

It should be noticed that the last functional dependency is induced by the formula `=I2*K2*0.2`, as described in section 3.1.1.

To compute the functional dependencies from the data we use a common data mining algorithm termed *Fun* (Novelli and Cicchetti, 2001). Note this algorithm is generic for finding functional dependencies in data. In the next subsections we will explain how to adapt it to get better results for spreadsheets.

3.1.1 Functional Dependencies From Formulas

Spreadsheets use formulas to define the values of some elements in terms of other elements. For example, in the running example, the values in column `M` (*profit*) are computed by multiplying the values in column `I` (*move*) by the values in column `K` (*price*) by 0.2; this is written as `M2 = I2 × K2 × 0.2` (for the second row). This formula states that the values in columns `I` (*move*) and `K` (*price*) determine the values in column `M` (*profit*), thus inducing the following functional dependency: *move price* $\rightharpoonup$ *profit* ($I\ K \rightharpoonup M$).

In general, formulas can have references to other cells also containing formulas. Consider, as an example, the formula `A1 = B1 * C1` in column `A`, and the formula `C1 = D1 + E1` in column `C`. Note these formulas define only the first row of a spreadsheet and that their row reference increments naturally throughout the rest of the rows. Because `C` is defined by another formula, the values that determine `C` also determine `A`. As a result, the two formulas induce the following functional dependencies:

$$B\ C \rightharpoonup A$$
$$D\ E \rightharpoonup C$$

Expanding the relationship between formulas, we get the following result:

$$B\ D\ E \rightharpoonup A$$
$$D\ E \qquad \rightharpoonup C$$

In general a spreadsheet formula of the 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 dependency explicit in all the elements. This may not always be the case and some elements can be defined otherwise (for example, by using a constant value or a different formula). In this case, no functional dependency is induced.

Functional dependencies induced by formulas are added to the ones computed by the *Fun* algorithm (Cunha et al, 2009b).

3.1.2 Filtering Functional Dependencies

The next step in this process is to filter out as many of the accidental dependencies as possible and keep the ones that are indicative of the underlying model. These can then be used to infer tables with primary and foreign keys.

The process of identifying the "valid" functional dependencies is, of course, ambiguous in general. Therefore, we employ a series of heuristics for evaluating dependencies.

Note that several of these heuristics are possible only in the context of spreadsheets. This observation supports the contention that end-user software engineering can benefit greatly from the context information that is available in a specific end-user programming domain (Erwig, 2009).

In the spreadsheet domain rich context is provided, in particular, through the spatial arrangement of cells and through labels (Erwig, 2009).

In the following we describe the employed heuristics. Each of these can add support to a functional dependency.

Label semantics. This heuristic is used to classify antecedents in functional dependencies. Most keys are labeled as "code" or "number" or are a combination of these labels with a label more related to the subject. For example, in a spreadsheet to store movies, a label "movie_id" could exist to uniquely identify a movie. We consider labels "id", "code", "number", "nr", "no" and combinations of them with other labels. A functional dependency with an antecedent of this kind receives high support.

Label arrangement. If the functional dependency respects the original order of the attributes, this counts in favor of this dependency since very often key attributes appear to the left of non-key attributes.

Antecedent size. Good primary keys often consist of a small number of attributes, that is, they are based on small antecedent sets. Therefore, the smaller the number of antecedent attributes, the higher the support for the functional dependency.

Ratio between antecedent and consequent sizes. In general, functional dependencies with smaller antecedents and larger consequents are stronger and thus more likely to be a reflection of the underlying data model. Therefore, a functional dependency receives the more support, the smaller the ratio of the number of consequent attributes is compared to the number of antecedent attributes.

Single value columns. It sometimes happens that spreadsheets have columns that contain just one and the same value. In our example, the columns *ok* and *qty* are like this. Such columns tend to appear in almost every functional dependency's consequent, which causes them to be repeated in many relations. Since in almost all cases, such dependencies are simply a consequence of the limited data (or represent redundant data entries), they are most likely not part of the underlying data model and will thus be ignored.

However, each of these columns is mapped to a functional dependency with the antecedent being the column name and the consequent an empty set. These dependencies are sound (Maier, 1983) and can be seen as representing relational tables with one attribute which is also the primary key.

After having gathered support through these heuristics, we aggregate the support for each functional dependency and sort them from most to least support. We then select functional dependencies from that list in the order of their support until all the attributes of the schema are covered.

Based on these heuristics, our algorithm produces the following dependencies for the Detergent application:

$$
\begin{aligned}
ok &\rightharpoonup \emptyset \\
qty &\rightharpoonup \emptyset \\
store\ week &\rightharpoonup onsale \\
move\ price &\rightharpoonup profit \\
com\_code &\rightharpoonup upc\ description\ size\ case\ nitem
\end{aligned}
$$

Note the first two functional dependencies are the ones created based on the *single value columns* heuristic.

One of the objectives of creating a relational model is to avoid update anomalies. To eliminate such problems, the dependencies must be normalized, for which we use an algorithm presented by Maier in (Maier, 1983).

Table 1 Foreign keys for our example.

Candidate Key Attributes		Foreign Key Attributes	
Schema	Attribute	Schema	Attribute
ConfirmationCode	ok	Sale	ok
Quantity	qty	Sale	qty
OnSale	store	Sale	store
OnSale	week	Sale	week
Profit	move	Sale	move
Profit	price	Sale	price
Detergent	com_code	Sale	com_code

Skipping over the details of that algorithm, we just note that in our example we obtain tables with only one candidate key each as shown in Table 1.
The final relational schema that we obtain is as follows.

$ConfirmationCode$ ($\underline{ok}$)
$Quantity$ ($\underline{qty}$)
$OnSale$ ($\underline{store}$, $\underline{week}$, $onsale$)
$Profit$ ($\underline{move}$, $\underline{price}$, $profit$)
$Detergent$ ($\underline{com\_code}$, upc, $description$, $size$, $case$, $nitem$)
$Sale$ ($\#com\_code$, $\#move$, $\#price$, $\#store$, $\#week$, $\#qty$, $\#ok$)

Our tool currently assigns single letters as table names. These can then be changed by the user (as we have done in this example). The complete technique is shown in Algorithm 1, which we term SPREADFD.

Algorithm 1 SPREADFD: Algorithm to calculate functional dependencies for spreadsheets.

---

input: a spreadsheet s
output: a set of sets of functional dependencies
  schemasAndRelations = findSchemaAndRelation s
  for all (schema, relation) in schemasAndRelations **do**
    onesAtts = get1col (schema, relation)
    oneFDs = map (λ att $\rightarrow$ (att, $\emptyset$)) onesAtts
    formulaAtts = findFormulaCols schema relation
    formulaFDs = createFormulaFDs formulaAtts schema relation
    f = fun (onesAtts $\cup$ formulaAtts) schema relation
    g = filter (f $\cup$ onesFDs $\cup$ formulaFDs) schema relation
    h = synthesize relation (lossless?) g
    j = selectKeys h
    k = toFDs j
  end for
  return set of functional dependencies k

For more detail about this algorithm the reader is referred to (Cunha, 2011).

3.2 The Relational Intermediate Directed Graph

The next step in the reverse engineering process is to produce a *Relational Intermediate Directed (RID) Graph* (Alhajj, 2003). This graph includes all the relationships between a given set of schemas. Nodes in the graph represent schemas and directed edges represent foreign keys between those schemas. For each schema, a node in the graph is created, and for each foreign key, an edge with cardinality "*" at both ends is added to the graph.
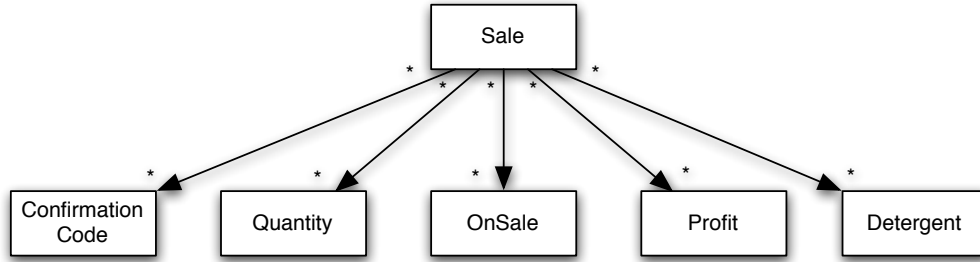
Fig. 6 RID graph for our running example.

Figure 6 represents the RID graph for the Detergent sales system. This graph can generally be improved in several ways. For example, the information about foreign keys may lead to additional links in the RID graph. If two relations reference each other, their relationship is said to be *symmetric* (Alhajj, 2003). One of the foreign keys can then be removed. In our example there are no symmetric references.

Another improvement to the RID graph is the detection of relationships, that is, whether a schema is a relationship connecting other schemas. In such cases, the schema is transformed into a relationship. The details of this algorithm are not so important and left out for brevity.

Since the only candidate key of the schema *Sale* is the combination of all the other schemas' primary keys, it is a relationship between all the other schemas and is therefore transformed into a relationship. The improved RID graph can be seen in Figure 7.
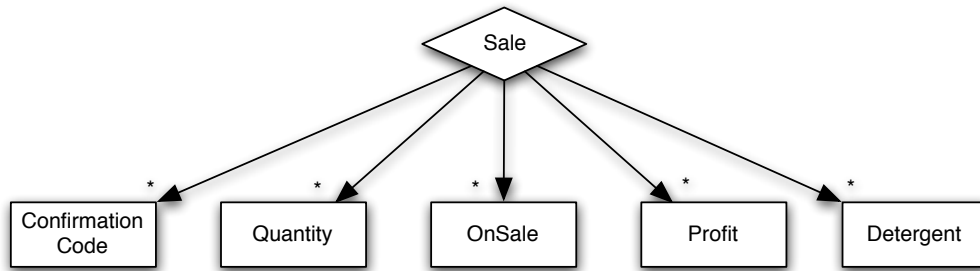

Fig. 7 Refactored RID graph.

This graph represents the relational schema we infer for our running example. Each of the rectangular nodes is a relation (the attributes are omitted), and the diamond represents the only relationship of the model. The links between the relationship and the other nodes represent the existence of a foreign key of each relation in the relationship. Finally, the asterisks represent the arity of the relationships. In this case the relationships are $M : N$. For more formal definitions we refer to (Alhajj, 2003).

We will further use this RID/relational schema to construct a ClassSheet to represent a spreadsheet, as explained in Section 6.

In the following section we will show that the relational schemas our tool can automatically infer are comparable in quality to the ones created by professional programmers.

4 Evaluation with Users

In this section we present a study performed to assess the quality of our automatic inference algorithm, SPREADFD, compared to expert subjects.

To assess our technique, we compare its results with the models produced by database experts. The main goal is to judge the effectiveness of our automatic model inference system. Moreover, we want to understand how experts extract a model from the data in a spreadsheet. This information can be used to improve our technique and tool. More formally, we seek to answer the following research questions:

RQ1 How well does SpreadFD perform compared to expert subjects in extracting relational models from spreadsheets?
RQ2 Which elements are used by participants when they are trying to understand spreadsheet data in order to develop their models?
RQ3 How important or relevant are the heuristics used by SpreadFD, given what the humans said they did?

4.1 Study Design

Our study aimed to answer if participants were able to perform their tasks with more accuracy given the experimental environment. We used a within subjects design, where each participant received three spreadsheets from different domains. The name of the spreadsheet *properties* was intentionally not given to participants to assess how much influence such information would have on the model they would create. This spreadsheet was termed as *anonymous*. Participants were asked to design the best relational model they could.

4.1.1 Methodology

The participants were asked to work as quickly and accurately as possible, but were not given time limits. Since the order of the spreadsheets was randomized (to compensate for the learning effect), they were told that the participant sitting next to them might appear to be moving faster, but that some tasks were shorter than others (we were trying to avoid having the participants stressed thinking they were performing worst than others). Following the tasks they had a post-session questionnaire, which contained questions to assess how participants worked to get to the final models and how confident they were of their results. These questions are:

1. *I am confident that I correctly did the tasks for spreadsheets dishes/ projects/ anonymous* (these were three different questions);
2. *The order of columns helped me to perform the tasks*;
3. *The name of columns in the first row of dishes and projects helped me to perform the tasks*;
4. *I used the formulas in projects and anonymous*;
5. *I treated in a different way columns which contained the same value in all the rows*;
6. *Describe in one paragraph what you think is the objective of the spreadsheets dishes/ projects/ anonymous.*

This last question will be used to assess the participants understanding of the different spreadsheets. Correct answers could only be given by participants having understood the running spreadsheets.

4.1.2 Participants

The participants of the study were students from a Masters degree on Support Decision Systems, including the courses on Administration and Exploration of Advanced Databases, Data Warehousing Systems, Analytic Data Processing (OLAP) and Data Mining.[1] In total, eighteen participants finished the study, two of them being females and sixteen being males. The participants were between 21-24 years of age. All were students at the university working on their Masters degree.

---

[1] More information about the degree can be found in `http://mei.di.uminho.pt/?q=en/1213/ssd-uk`.

4.1.3 Study Tasks

Each participant received three images of three different spreadsheets, that is, a paper sheet with a picture of the spreadsheet. The first spreadsheet is called *dishes* and is shown in Figure 8. In fact, this is a similar spreadsheet to the one we are using as the running example. However, the one we gave to participants does not contain formulas. The name of the spreadsheet and an indication that it does not contain formulas was also given to participants.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	COM_CODE	UPC	DESCRIP	SIZE	CASE	NITEM	STORE	WEEK	MOVE	QTY	PRICE	SALE	PROFIT	OK
2	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	383	16	1	2.19		33.47	1.00
3	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	384	7	1	2.19		33.47	1.00
4	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	385	15	1	2.19		33.47	1.00
5	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	386	14	1	2.19		38.49	1.00
6	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	387	14	1	2.19		43.51	1.00
7	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	388	14	1	2.19	S	43.51	1.00
8	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	389	7	1	2.19		43.51	1.00
9	653.00	1111140009	DOVE DISH LIQUID	42 OZ	9.00	2851281	100	390	12	1	2.19		43.51	1.00
10	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	387	0	1	0.00		0.00	1.00
11	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	388	0	1	0.00		0.00	1.00
12	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	389	0	1	0.00		0.00	1.00
13	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	390	0	1	0.00		0.00	1.00
14	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	391	0	1	0.00		0.00	1.00
15	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	102	392	0	1	0.00		0.00	1.00
16	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	103	383	0	1	0.00		0.00	1.00
17	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	105	390	0	1	0.00		0.00	1.00
18	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	105	391	0	1	0.00		0.00	1.00
19	653.00	1111140188	DOVE DISH LIQUID PP$	42 OZ	9.00	2851281	105	392	0	1	0.00		0.00	1.00
20	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	383	0	1	0.00		0.00	1.00
21	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	384	0	1	0.00		0.00	1.00
22	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	385	0	1	0.00		0.00	1.00
23	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	386	0	1	0.00		0.00	1.00
24	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	387	0	1	0.00		0.00	1.00
25	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	388	0	1	0.00		0.00	1.00
26	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	389	0	1	0.00		0.00	1.00
27	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	390	0	1	0.00		0.00	1.00
28	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	391	0	1	0.00		0.00	1.00
29	653.00	1111140204	DOVE LIQUID	128 OZ	4.00	2825021	101	392	0	1	0.00		0.00	1.00
30	653.00	1111140311	DOVE DISH LIQUID	22 OZ	16.00	2851241	100	383	19	1	1.19		29.41	1.00
31	653.00	1111140311	DOVE DISH LIQUID	22 OZ	16.00	2851241	100	384	18	1	1.16		28.06	1.00
32	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	383	9	1	2.99		21.97	1.00
33	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	384	21	1	2.99		21.97	1.00
34	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	385	10	1	2.99		21.97	1.00
35	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	386	6	1	2.99		21.97	1.00
36	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	387	5	1	2.99		21.97	1.00
37	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	388	5	1	2.99		21.97	1.00
38	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	389	13	1	2.99		21.97	1.00
39	654.00	1111143002	SUNLIGHT DISH LIQUID	64 OZ	6.00	2856991	100	390	6	1	2.99		21.97	1.00

Fig. 8 A spreadsheet representing a dishwasher detergents selling system.

The second spreadsheet they received is named *projects* and is shown in Figure 9.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	project_nr	manager	location	delivery_date	budget	institute_name	employee_name	age	nationality	empl_supervisor	instrument_nr	capacity	nr_of_wheels
2	proj1	John	New York	30-03-2010	50000	Long Island	Richy	34	USA	Mike	inst3	36	6
3	proj1	John	New York	30-03-2010	50000	Long Island	Tim	33	JP	Anthony	inst1	24	4
4	proj1	John	New York	30-03-2010	50000	Long Island	Mark	30	UK	Alfred	inst3	36	6
5	proj2	John	Los Angels	02-04-2010	3000	Los Angels	Richy	34	USA	Mike	inst2	30	5
6	proj3	Paul	Chicago	01-01-2009	12000	Chicago	Tim	33	JP	Anthony	inst1	24	4
7	proj3	Paul	Chicago	01-01-2009	12000	Chicago	Mark	30	UK	Alfred	inst1	24	4
8	proj4	Mike	San Francisco	15-04-2008	4000	Bay Institute	Richy	34	USA	Mike	inst3	36	6
9	proj5	James	Savannah	15-04-2008	15000	Savannah	Tim	33	JP	Anthony	inst1	24	4
10	proj5	James	Savannah	15-04-2008	15000	Savannah	Mark	30	UK	Alfred	inst3	36	6
11	proj5	James	Savannah	15-04-2008	15000	Savannah	Richy	34	USA	Mike	inst2	30	5
12	proj6	John	Chicago	01-04-2010	12000	Chicago	Tim	33	JP	Anthony	inst1	24	4
13	proj6	John	Chicago	01-04-2010	12000	Chicago	Mark	30	UK	Alfred	inst1	24	4
14	proj7	Raul	Seattle	12-04-2010	21000	Seattle	Jone	42	USA	Michael	inst2	30	5
15	proj7	Raul	Seattle	12-04-2010	21000	Seattle	Tim	33	JP	Anthony	inst1	24	4

Fig. 9 A spreadsheet representing project management system.

This spreadsheet implements a system to manage projects, storing information about projects, workers, and instruments used. It was adapted from (Alhajj, 2003). The name of the spreadsheet and a description of the formula in column L was also given to participants. The description was given as follows:

Column L is calculated according to the formula $L_i = M_i * 6$.

When doubts appeared because of the formulas, they were explained.
The last spreadsheet is called *properties* and is presented in Figure 10.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	cr76	pg4	John	4445454	6 Lawrence	UK	01-07-2000	31-08-2001	426	50	21300	co40	Tina	tina@gmail.com
3	cr76	pg16	John	4445454	5 Nuvar Dr.	UK	01-09-2001	01-09-2002	365	70	25550	co93	Tony	tony@aol.com
4	cr56	pg4	Aline	5552312	6 Lawrence	UK	01-09-1999	10-06-2000	283	50	14150	co40	Tina	tina@gmail.com
5	cr56	pg36	Aline	5552312	2 Manor Rd	UK	10-10-2000	01-09-2001	326	60	19560	co12	Anne	anne@gmail.com
6	cr56	pg16	Aline	5552312	5 Nuvar Dr.	UK	01-11-2002	10-10-2003	343	70	24010	co93	Tony	tony@aol.com
7	cr13	pg16	Johnny	4443465	5 Nuvar Dr.	UK	03-01-2008	01-11-2009	668	70	46760	co93	Tony	tony@aol.com
8	cr13	pg12	Johnny	4443465	22 5ᵗʰ Avenue	UK	01-01-2010	03-01-2010	2	35	70	co33	July	july@gmail.com
9	cr76	pg19	John	4445454	13 Sea Av.	UK	30-01-2006	01-01-2010	1432	40	57280	co12	Anne	anne@gmail.com
10	cr56	pg4	Aline	5552312	6 Lawrence	UK	09-09-2008	01-09-2009	357	50	17850	co40	Tina	tina@gmail.com
11	cr24	pg16	Mike	3337688	5 Nuvar Dr.	UK	03-03-2000	10-10-2000	221	70	15470	co93	Tony	tony@aol.com
12	cr13	pg4	Johnny	4443465	6 Lawrence	UK	02-11-2001	01-11-2002	364	50	18200	co40	Tina	tina@gmail.com
13	cr13	pg16	Johnny	4443465	5 Nuvar Dr.	UK	13-06-2002	03-01-2008	2030	70	142100	co93	Tony	tony@aol.com
14	cr76	pg4	John	4445454	6 Lawrence	UK	16-05-2002	01-01-2010	2787	50	139350	co40	Tina	tina@gmail.com
15	cr56	pg36	Aline	5552312	2 Manor Rd	UK	04-04-2005	01-09-2006	515	60	30900	co12	Anne	anne@gmail.com
16	cr24	pg16	Mike	3337688	5 Nuvar Dr.	UK	01-09-2001	10-06-2002	282	70	19740	co93	Tony	tony@aol.com
17	cr47	pg12	Joshua	5554786	22 5ᵗʰ Avenue	UK	02-11-2001	01-02-2008	2282	35	79870	co33	July	july@gmail.com
18	cr13	pg19	Johnny	4443465	13 Sea Av.	UK	09-09-2009	01-02-2010	145	40	5800	co12	Anne	anne@gmail.com
19	cr13	pg4	Johnny	4443465	6 Lawrence	UK	30-01-2006	10-10-2007	618	50	30900	co40	Tina	tina@gmail.com
20	cr55	pg16	Raul	1112319	5 Nuvar Dr.	UK	02-11-2001	01-11-2002	364	70	25480	co93	Tony	tony@aol.com
21	cr13	pg4	Johnny	4443465	6 Lawrence	UK	16-05-2002	03-01-2008	2058	50	102900	co40	Tina	tina@gmail.com
22	cr13	pg36	Johnny	4443465	2 Manor Rd	UK	02-11-2001	01-09-2010	3225	60	193500	co12	Anne	anne@gmail.com
23	cr55	pg16	Raul	1112319	5 Nuvar Dr.	UK	02-11-2001	10-10-2002	342	70	23940	co93	Tony	tony@aol.com
24	cr13	pg12	Johnny	4443465	22 5ᵗʰ Avenue	UK	16-05-2002	01-11-2002	169	35	5915	co33	July	july@gmail.com
25	cr24	pg19	Mike	3337688	13 Sea Av.	UK	04-04-2005	03-01-2008	1004	40	40160	co12	Anne	anne@gmail.com
26	cr47	pg4	Joshua	5554786	6 Lawrence	UK	09-09-2009	01-01-2010	114	50	5700	co40	Tina	tina@gmail.com
27	cr47	pg16	Joshua	5554786	5 Nuvar Dr.	UK	04-04-2005	09-04-2010	1831	70	128170	co93	Tony	tony@aol.com

Fig. 10 A spreadsheet representing a properties renting system.

This spreadsheet represents a properties renting system, storing information about renters, properties, their owners, and about the renting actions themselves. This spreadsheet was adapted from (Connolly and Begg, 2002). Notice that the first row in the spreadsheet is empty, that is, the labels of columns were not given to the participants. In fact, the name of the spreadsheet was also omitted. The purpose was to understand weather the labels help participants to understand the spreadsheet. The description of the formulas used by the spreadsheet was given:

Column I is calculated according to the formula $I_i = H_i - G_i$. Column K is calculated according to the formula $K_i = J_i * L_i$.

Grading. In the end of the study, participants answered a questionnaire which was collected together with all models they created. We also answered a questionnaire and wrote our models based on the result of SPREADFD in such a way that it could not be possible to identify it. Thus, all the questionnaires where anonymous. Each model was graded by a professor specialized in the area of databases, Professor Orlando Belo. The grades range between 0 and 5 points.

4.2 Threats to Validity

As suggested by Perry et al. (Perry et al, 2000), we discuss three types of influences that might limit the validity of our study.

4.2.1 Construct Validity

i) Participants expertise: The participants of our study are all Master students in the area of Support Decision Systems having to follow several courses on advanced databases. Moreover, all the students already had followed a course on databases as undergraduate students. Their final mark average of 9.9 (in a 0 to 15 scale) shows that they are good students. These facts show that we can in fact consider our participants as database experts, as also been described in (Höst et al, 2000).

4.2.2 Internal Validity

i) Accuracy of grades: Each model was graded by a professor specialized in the area of databases. He has a large experience (more than twenty years in the area) as a professor, researcher and working with real problems in industry. He was/is responsible for several courses on databases and has a large experience grading similar works. Even if the grades are not perfect since he did not know which models where generated by our tool, he could never favor our tool. Thus, the grades are uniform for all the models.

4.2.3 External Validity

i) Generalization: In this study we used three different spreadsheets from different domains. We believe that the results can be generalized to other spreadsheets. The models our tool can generate 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. In fact, we will show in Section 5 our algorithm can be applied to a wide range of spreadsheets.

ii) Industrial usage: The spreadsheets we used are adapted from other research works and not from industry. Nevertheless, our tool is not specific to these spreadsheets and can be used with any spreadsheet. The models we produce are of good quality, even when compared to experts, as we will see in Section 4.3. Thus, we believe our tool can be used in a professional environment, such as the one found in industry.

4.3 Results

In Table 2 we present the results of our study. The first column contains each participant, including the results of SPREADFD (number 8), the three middle columns present the grade for each different spreadsheet and the last column shows the final score for each participant. The second but last row shows the average for each of the different spreadsheets.

Table 2 Results of our evaluation with users.

participant	dishes	projects	properties	total
1	3	2	5	10
2	4	5	4	13
3	4	4	3	11
4	4	3	1	8
5	4	3	2	9
6	4	2	4	10
7	1	2	2	5
9	4	5	2	11
10	4	4	2	10
11	2	5	1	8
12	5	5	3	13
13	2	4	4	10
14	4	4	2	10
15	2	2	3	7
16	5	5	3	13
17	4	4	3	11
18	4	2	5	11
19	3	4	2	9
average	3.5	3.6	2.8	9.9
8 (SPREADFD)	3	5	2	10

From these results we can notice that only two participants received a perfect score in the *dishes* and *properties* spreadsheets while in *projects* five participants were perfect. On the other hand, no participant was graded with 0, but in *dishes* one participant received 1 as the score, and in properties two were graded with 1 value. Moreover, we can see that in *dishes* and in *properties* the tool is below the average, but in *projects* it is above. In the final grade, SPREADFD is very close to the average.

A two sample Wilcoxon rank sum test was performed (R implementation) using the collected data from the study. The alternative hypothesis stated a one-sided test. The distribution representing the performance of the best user (participant 12 in Table 2) and our system (row 8) give no significant difference for a standard alpha=0.05. However, this result in not statistically significant as the p-value is 0.2398. This suggests that the quality of the proposed models are comparable to the ones yielded by a human expert. Global average performance shown in Figure 11 also supports these conclusions.
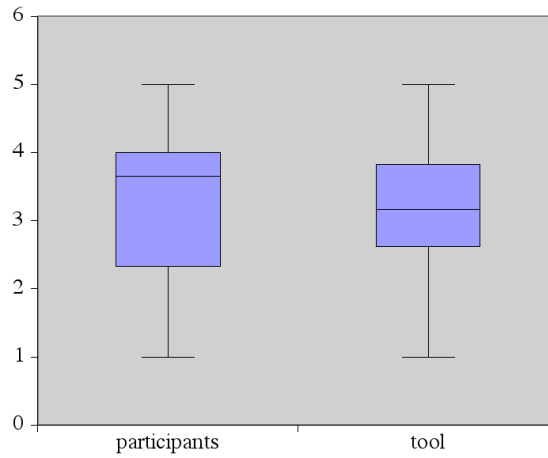
Fig. 11 Boxplots of the scores both for the study participants and our tool.

4.4 Answering the Research Questions

In this section we answer the research questions we proposed in the beginning of this study.

RQ1 How well does SPREADFD perform compared to expert subjects in extracting relational models from spreadsheets?

For the *dishes* spreadsheet our tool was marked with 3 points. The average of participants for this spreadsheet was 3.5. This means that we are slightly bellow the average. Nevertheless, this is a positive result for an automatic tool. For the *projects* spreadsheet SPREADFD received 5 points. The average for this spreadsheet was 3.6 which means we are way above it. In fact, the tool got the maximum grade in this spreadsheet. Only 5 participants reached this level. Finally, for the *properties* spreadsheet the tool was marked with 2 points. The average for this spreadsheet was 2.8 and in fact SPREADFD is not far from it. Given the marks for this spreadsheet, it seems that it is the most difficult one from our set. As a final mark, summing all the grades, SPREADFD got 10 points in 15 possible while the average of participants was 9.9.

From these observations, we conclude that SPREADFD can produce models of the same quality that expert users would produce.

RQ2 Which elements are used by participants when they are trying to understand spreadsheet data in order to develop their models?

To answer this question we analyzed the questionnaires participants filled out. Table 3 summarizes the answers related to the use of certain patterns to create the models. The first column contains the possible values of the participants' answers; the next four columns contain the number of participants and their concordance on the use of the order of columns to create the model, the use of labels, the use of formulas, and the use of columns with the same value in all rows, respectively. The last column sums all other columns. Please refer to Section 4.1.1 for the exact questions.

concordance / question	order of columns	use labels	use formulas	columns w/ same value	total
completely disagree	2	0	1	1	4
disagree	5	2	1	0	8
neither agree nor disagree	5	5	1	2	13
agree	6	8	11	11	36
completely agree	0	3	4	4	11

Table 3 Answers given by participants about the use of patterns to create the models.

From these results we can see that participants clearly used the columns with the same value in all rows and formulas in a special way to create the models since 15 (out of 19) answered positively in each of those patterns. As for labels, we can see that 11 responded positively to their use in the creation of models. This means that most of the study participants used the labels. Indeed, only 3 did not use them completely. The order of columns seems to be the less used heuristic. Still, 6 of the study participants used such knowledge. If we look at totals, we can see that 65.3% of all answers (47 out of 72) agree with the fact that some heuristic was used. In fact, only 16.7% of all answers (12 out of 72) did not used heuristics at all. Moreover, the average of marks for the *anonymous* spreadsheet is the worst and we only removed the labels from it. From these facts we can see that participants used most of the heuristics our tool also uses.

RQ3 How important or relevant are the heuristics used by SpreadFD, given what the humans said they did?

To answer this last research question we looked into the participants' answers to the post-questionnaire. According to the answers to the post-questionnaire the only heuristic which participants did not agree on its usefulness is the column order: only 6 out of 18 used this heuristic to compute the model (see Table 3). Since most students computed a realistic model without needing this heuristic, the question that arises is whether is it useful for our algorithm or not. Thus, we removed it from SpreadFD and ran it again with the examples from the study. The models generated are indeed the same as the ones produced using this heuristic. As a consequence we can say that for this particular examples the heuristic is not relevant: it did not improve nor damaged the computed models. For other spreadsheets this heuristic does influence positively the inference of the model. A detailed study should be conducted to conclude whether this heuristic should be part of our algorithm or not.

5 Applicability Evaluation

In order to evaluate the applicability of our approach, we have performed an experiment on the spreadsheets that are made available (through a CD) with (Powell and Baker, 2003). This set consists of 27 spreadsheets, each containing between 1 and 16 worksheets, with a total of 121 worksheets[2]. More than half of the worksheets, 66 out of the 121, contain formulas. The spreadsheets represent different realms that range from spreadsheets to make the decision of how much to bid for the salvage rights to a grounded ship, modeling the production at Delta Oil, advertising budget, or planning shipments.

---

[2] One spreadsheet, c6/Adbudget6.xls, was unreadable.

With this experiment we want to test whether the inference approach we propose works in practice. Specifically, we want to know how well the system is able to identify table and relationship structures and for which kinds of spreadsheets it works and when it fails. We also want to know the quality of the relational models generated.

More precisely, we seek to answer the following research questions:

RQ1: In how many cases is relational model inference applicable?

RQ2: How many of the table and relationship structures that can be identified in the data can be successfully captured by relational models inferred by our tool?

RQ3: In which cases does model inference fail?

5.1 Test Results

To answer the first two research questions we manually inspected all the spreadsheets to see how many tables could be identified and what relationships exist.

We present in Table 4 the results of this evaluation. In the first column we list the name of the spreadsheets used, in the second/third column we count the number of sheets and tables in each spreadsheet, the fourth column contains the number of tables that the tool could not identify, and the last three columns list the classification we gave to the models produced by our technique (bad, acceptable, and good models, respectively). Four of the tables in spreadsheet "c9/Options.xls" contained spreadsheet errors (as reported by Excel); these tables were excluded from further analysis.

Table 4 Results of our evaluation.

Spreadsheet	worksheets	tables	fail	bad	acceptable	good
c4/SS Kuniang.xls	1	2				2
c5/AdBudget.xls	8	13	6			7
c5/Delta.xls	5	5			2	3
c7/Bundy.xls	10	44			2	42
c7/Forecasting.xls	7	10	6			4
c7/Analgesics.xls	3	5		1		4
c7/Applicants.xls	6	6		2	1	3
c7/Dish.xls	2	2				2
c7/Executives.xls	6	11		2	1	8
c7/Population.xls	3	4			4	
c7/Tissue.xls	1	1			1	
c8/AdBudget8.xls	8	8		2	2	4
c8/IP.xls	7	1		1		
c8/LP.xls	16	7			2	5
c8/NLP.xls	5	5				5
c9/AdBudget9.xls	7	6		1	1	4
c9/Butson.xls	2	4			4	
c9/Data.xls	2	10				10
c9/Diffusion.xls	2	4		1		3
c9/Hastings1.xls	3	4		2		2
c9/Hastings2.xls	2	1				1
c9/Netscape.xls	5	8			7	1
c9/Options.xls	8	4	1			3
c9/Plants.xls	2	10				10
c9/Portfolio.xls	3	1				1
c9/Veerman.xls	1	0				
Total	121	176	13	12	27	124

Through manual inspection of the spreadsheets, we were able to identify 176 tables. In the best case, the inference would be able to identify all 176 tables and create a relational model representation for each of them.

The results in Table 4 show that the tool could identify all but 13 tables. The tool failed mainly in those cases where no layout in the spreadsheets was available. Although through manual inspection we could recognize a table structure, the tool was unable to derive sufficient constraints from layout to support the heuristics for the successful detection of functional dependencies. Note that the heuristics used are the same for all the spreadsheets.

Inspection of the 163 successfully produced models suggested that they should be classified into three different levels of quality: *bad*, *acceptable*, and *good*.

A relational model is classified as *bad* if it is not realistic. In some cases it is not possible to infer a model that is similar to the one an expert would create. Although from a data point of view it is correct and normalized, we consider that an expert would produce a better model. We found that 12 models fall under this classification.

The classification *acceptable* was given to models that do not completely characterize the corresponding table or relationship; while capturing many or most of its essential aspects, it left out some important parts. We classified 27 relational models as *acceptable*.

Finally, a *good* relational model is a model that closely represents the tables and relationships under consideration, thus being very realistic. From the 163 tables that model inference was able to process, the tool produced 124 good relational models.

5.2 Discussion

The test results are quite encouraging: With our techniques we are able to produce relational models for more than 92% of the existing tables. Of these models, more than 76% are classified as good.

The failure to generate good models in about 1/4th of the cases was mostly due to two facts: (1) lack of layout to inform the heuristics and (2) some dependencies that do hold for the models did not appear in the data.

Although our process is completely automatic, we believe that the above observations point to the fact that the method could be much more effective if it was guided by a human. For example, if additional dependencies or headers were explicitly provided by the user, the generated models could be improved. Therefore, an extension of our tool allowing users to interact and provide input about the models seems to be the most important direction for future work.

6 Mapping Relational Models to ClassSheets

In previous sections we have introduced an algorithm to generate relational models from existing spreadsheets. Although these models are widely used to represent data, they do not consider any layout restrictions, which is a very important feature of spreadsheets. ClassSheets however were designed to model spreadsheets, including its layout. Thus, in this section we introduce a mapping from relational models to ClassSheets.

6.1 From Relational Models to ClassSheets

The relational models generated in Section 3.2 (in the form of a RID graph) can be directly translated into a ClassSheet diagram. By default, each node is translated into a class with the same name as the relation and a vertically expanding block. In general, for a relation of the form

$$\underline{A_1}, \ldots, \underline{A_n}, A_{n+1}, ..., A_m$$

and default values $da_1, \ldots, da_n, d_{n+1}, ..., d_m$, a ClassSheet class/table is generated[3] as shown in Figure 12. Note that the relation may be composed only of key attributes, as commonly happens in database relations. From now on this rule is termed *rule 1*.

---

[3] We omit here the column labels, whose names depend on the number of columns in the generated table.

Fig. 12 Generated class for a relation A.

This ClassSheet represents a spreadsheet "table" with name **A**. For each attribute, a column is created and is labeled with the attribute's name. The default values depend on the attribute's domain. This table expands vertically, as indicated by the ellipses. The key attributes become underlined labels.

A special case occurs when there is a foreign key from one relation to another. The two relations are created basically as described above but the attributes that compose the foreign key do not have default values, but references to the corresponding attributes in the other class. Let us use the following generic relations:

$$M(\underline{M_1, ..., M_r}, M_{r+1}, ..., M_s)$$
$$N(\underline{N_1, ..., N_t, M_n, ..., M_m, M_o, ..., M_p}, N_{t+1}, ..., N_u)$$

Note that $M_n, ..., M_m, M_o, ..., M_p$ are foreign keys from the relation N to the relation M, where $1 \leqslant n, m, o, p \leqslant r$, $n \leqslant m$, and $o \leqslant p$. This means that the foreign key attributes in N can only reference key attributes in the M. Again, it may happen that the relations have only key attributes. The corresponding ClassSheet is illustrated in Figure 13. This rule is termed *rule 2*.

Fig. 13 Generated ClassSheet for relations with foreign keys.

Relationships are treated differently and will be translated into cell classes. We distinguish between two cases: *(A)* relationships between two schemas, and *(B)* relationships between more than two schemas.

For case *(A)*, let us consider the following set of schemas:

$$M(\underline{M_1, ..., M_r}, M_{r+1}, ..., M_s)$$
$$N(\underline{N_1, ..., N_t}, N_{t+1}, ..., N_u)$$
$$R(\underline{M_1, ..., M_r, N_1, ..., N_t, R_1, ..., R_x}, R_{x+1}, ..., R_y)$$

The ClassSheet that is produced by this translation is shown in Figure 14 and explained next.

For both nodes M and N a class is created as explained before (lower part of the ClassSheet). The top part of the ClassSheet is divided in two classes and one cell class. The first class, **NKey**, is created using the key attributes from the **N** class. All its values are references to **N**. For example, $n1 = \mathbf{N.N1}$ references the values in column **A** in class **N**. This makes the spreadsheet easier to maintain while avoiding insertion, modification and deletion anomalies(Codd, 1970). Class **Mkey** is created using the key attributes of the class **M** and the rest of the key attributes of the relationship R. The cell class (with blue border) is created using the rest of the attributes of the relationship R.

	A										...
1	R			Mkey							
2				M_1	...	M_r	R_1	...	R_x		
3				$m_1 = M.M_1$	...	$m_r = M.M_r$	$r_1 = dr_1$	...	$r_x = dr_x$		
4	Nkey										
5	N_1	...	N_t	R_{x+1}	...	R_y					
6	$n_1 = N.N_1$	...	$n_t = N.N_t$	$r_{x+1} = dr_{x+1}$	...	$r_y = dr_y$					
7											
8	A										
9	N										
10	N_1	...	N_t	N_{t+1}	...	N_u					
11	$n_1 = dn_1$	...	$n_t = dn_t$	$n_{t+1} = dn_{t+1}$	...	$n_u = dn_u$					
12											
13	A										
14	M										
15	M_1	...	M_r	M_{r+1}	...	M_s					
16	$m_1 = dm_1$	...	$m_r = dm_r$	$m_{r+1} = dm_{r+1}$	...	$m_s = dm_s$					

Fig. 14 ClassSheet of a relationship connecting two relations.

The classes for M and N are always created, even if they have only key attributes. If classes were not created and the values were inserted in the top part of the spreadsheet, it could be necessary to insert the same values several times to represent their relationships with the values in N. Since the classes are always created, on the top part it is only necessary to refer to the values in M or N, thus avoiding data redundancy. This is also true for the two sub-cases of this rule, which are presented next.

In principle, the positions of **M** and **N** are interchangeable and we have to choose which one expands vertically and which one expands horizontally. We choose whichever combination minimizes the number of empty cells created by the cell class, that is, the number of key attributes from **M** and **R** should be similar to the number of non-key attributes of **R**. This rule is named *rule A*. Two special cases can occur with this configuration.

Case 1. The first case occurs when the key of the relationship R is only composed by the keys of M and N (defined as before), that is, R is defined as follows:

$$M(\underline{M_1, ..., M_r}, M_{r+1}, ..., M_s)$$
$$N(\underline{N_1, ..., N_t}, N_{t+1}, ..., N_u)$$
$$R(\underline{M_1, ..., M_r, N_1, ..., N_t}, R_1, ..., R_x)$$

The resultant ClassSheet is shown in Figure 15.

The difference between this ClassSheet model and the general one is that the **MKey** class on the top does not contain any attribute from R: All its attributes are contained in the cell class. This rule is from now on named *rule A1*.

Case 2. The second case occurs when the relationship is composed only by key attributes as illustrated next:

$$M(\underline{M_1, ..., M_r}, M_{r+1}, ..., M_s)$$
$$N(\underline{N_1, ..., N_t}, N_{t+1}, ..., N_u)$$
$$R(\underline{M_1, ..., M_r, N_1, ..., N_t})$$

In this situation, the relationship is constituted only by the key attributes of **M** and **N**. Thus, in the cell class there is no attributes nor data to include. However, there must be a way to effectively establish the relationship between each entry in **MKey** and **Nkey**. Thus, a simple

Fig. 15 ClassSheet of a relationship with all the key attributes being foreign keys.

attribute **related** (also labeled **Related**) is added to the cell class with default value 0. When adding values to **MKey** and **Nkey** the attribute **related** will receive a 0 meaning that there is no relationship between the two entries. If the user wants to express there is indeed such relation, then the value of **related** must be set to 1. Figure 16 illustrates this case. From now on this rule is named *rule A2*.

Fig. 16 ClassSheet of a relationship composed only by key attributes.

It is not possible to represent relationships with more than two tables in ClassSheets. Thus, for case *(B)*, that is, for relationships between more than two tables, we need to choose between the candidate tables to span the cell class. In this case we try to create the most visual appealing spreadsheet possible. To do so, we use the following criteria to decide which tables should be chosen to create the cell class:

1. the number of empty cells created by the cell class should be minimal;
2. the key of **N** (the class that expands vertically) should be minimal.

Both criteria are designed to compute a cell class similar to what a person would do and therefore it must avoid to create too many unused or blank cells.

More precisely, for any combination of **M**, **N**, and **R**, let us assume that $\underline{n}$ and $\underline{r}$ correspond to be the number of attributes composing the key of **N** and **R**, respectively, and r the number of non-key attributes in **R**.

To choose the three relations we find the ones that first minimize the following expression:

$$|(\underline{r} - \underline{n}) - max(1, r)|$$

This enforces the first criterion defined above by calculating the absolute value of the difference between the size of the keys of **R** and **N**, which will give us the number of columns in the cell class. From this we subtract $max(1, r)$, which is the number of columns that will not be empty in the cell class: 1 if **R** has no non-key attributes, in which case **related** is added, or r to count the number on non-key attributes of **R**, attributes that are placed in the cell class.

If ambiguities occur, then we choose the combination that minimizes the size of the key of **N** ($\underline{n}$).

Although the choice of **M** has no direct influence on the criteria defined above, it implicitly influences it as its key is included in the key of **R**.

This rule is from now on named *rule B*.

After having chosen the two relations (and the relationship), the generation proceeds as described above. The remaining relations are created as explained in the beginning of this section. In the next subsection we explain in more detail how to combine these rules to achieve the resulting ClassSheet.

6.2 Mapping Strategy

In this section we present the mapping function between RID graphs and ClassSheets, which builds on the rules presented before. For that, we use the common strategic combinators listed below (Lämmel and Visser, 2003; Visser and Saraiva, 2004; Visser, 2005):

$\triangleright$ $:: Rule \rightarrow Rule \rightarrow Rule$ -- sequential composition
$\oslash$ $:: Rule \rightarrow Rule \rightarrow Rule$ -- left-biased choice
$many :: Rule \rightarrow Rule$ -- repetition
$once :: Rule \rightarrow Rule$ -- arbitrary depth rule application

In this context, *Rule* encodes a transformation from RID graphs to ClassSheets.

Using the rules defined in the previous section and the combinators listed above, we can construct a strategy that generates a ClassSheet:

$genCS =$
    $many\ (once\ (rule\ B)) \triangleright$
    $many\ (once\ (rule\ A)) \triangleright$
    $many\ (once\ (rule\ A1)\ \oslash\ once\ (rule\ A2)) \triangleright$
    $many\ (once\ (rule\ 2)) \triangleright$
    $many\ (once\ (rule\ 1))$

The strategy works as follows: It tries to apply *rule B* as many times as possible, consuming all the relationships with more than two relations; it then tries to apply *rule A* as many times as possible, consuming relationships with two relations; next the two sub-cases of *rule A* are applied as many times as possible consuming all the relationships with two relations that match some of the sub-rules; after consuming all the relationships and corresponding relations, the strategy consumes all the relations that are connected through a foreign key using *rule 2*; finally, all the remaining relations are mapped using *rule 1*.

The ClassSheet model that is generated by our tool for the Detergent application can be see in Figure 2.

7 The MDSheet Framework

The inference techniques presented in previous sections have been implemented as a new feature of an existing office extension, named MDSheet. This tool can be included in a widely used spreadsheet system, namely OpenOffice/LibreOffice. It provides a model-driven spreadsheet development environment where a (model-driven) spreadsheet consists of two types of worksheets: `Sheet 0`, containing the embedded ClassSheet model, and `Sheet 1`, containing the spreadsheet data that conforms to the model. Users can interact both with the ClassSheet model and the spreadsheet data. Our techniques guarantee the synchronization of the two representations (Cunha et al, 2011a,b).

The model inference is particularly interesting when the spreadsheet under consideration does not have a corresponding model. Our framework offers a button on its interface that when pressed automatically generates the ClassSheet using the techniques presented in previous sections. The MDSheet framework allows also the development of ClassSheet models from scratch by using visual objects (for example, buttons, pull-down menus) or by traditional editing. When the ClassSheet is defined, a first instance/spreadsheet is generated (from the ClassSheet). Such a spreadsheet embeds the defined/inferred business logic rules as regular spreadsheet formulas which assist end users in the safe and correct inputting/editing of the spreadsheet data. The embedding of business logic rules as regular spreadsheet formulas is also implemented when inferring the ClassSheet model from an existing spreadsheet: The original spreadsheet data is automatically refactored so that it provides a similar editing assistance.

The global architecture of the model-driven spreadsheet development framework we have constructed is presented in Figure 17.
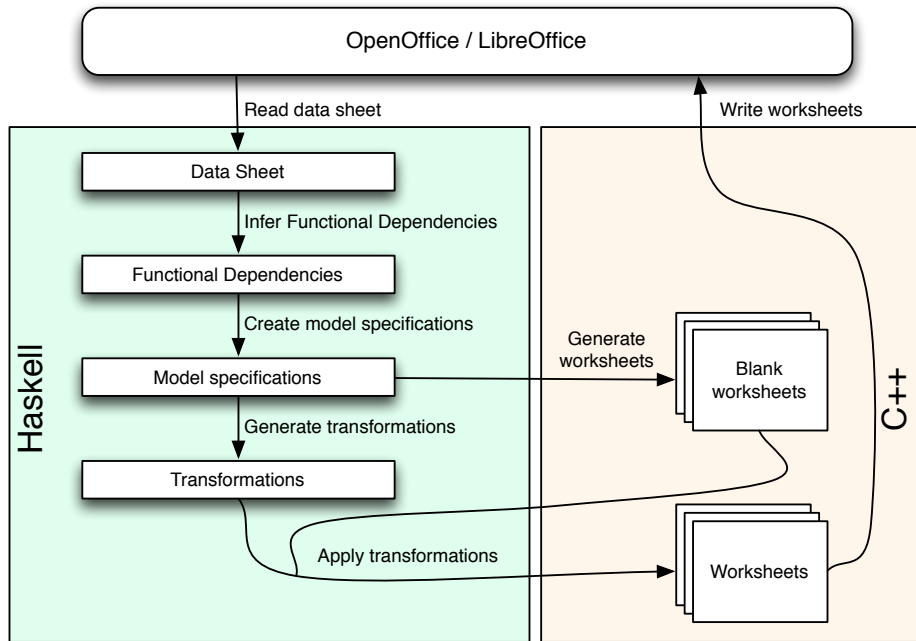

Fig. 17 Architecture of MDSheet with automatic model inference.

Having a model-driven spreadsheet development environment, end users can not only edit the spreadsheet data (conforming to the model), but they can also evolve the model. Our environment automatically ensures the synchronization of the ClassSheet model and the spreadsheet data. The evolution of the model/instance may induce different functional dependencies on the spreadsheet data. Our inference algorithm guarantees the synchronization after such an evolution step.

The MDSheet framework and a video with a demonstration of its capabilities are available at `http://ssaapp.di.uminho.pt`.

In the next section we present in detail the study we have organized and conducted to assess the applicability of automatic model inference.

8 Related Work

In previous work we introduced the concept of inferring ClassSheets from spreadsheet data (Cunha et al, 2010). This inference technique is revisited in the first part of this paper. The second part of the paper presents an empirical evaluation of such technique. We organized and ran an empirical study which showed empirical evidences that the results computed by our technique are comparable to the results produced by databases experts. Moreover, in (Cunha et al, 2014a) we also used functional dependencies to improve spreadsheets. The devised functional dependencies are used to add visual objects to the spreadsheets, which help the users to input/edit data. They are also used to refactor the original spreadsheet into a new, normalized one, without data redundancy and the related data problems. In this new work we present the inference algorithm for the first time. Moreover, we introduce more heuristics, which help to infer better relational models.

In (Cunha et al, 2011a) we have presented the embedding of ClassSheets in spreadsheet systems. We have reused the available tools of OpenOffice to embed the visual representation of ClassSheets in a general purpose system. This embedding allows to have a model and its instance in the same environment, the one the user already knows and is familiar with. Moreover, in this way it is possible to keep both the model and data synchronized (Cunha et al, 2012a,b). In the same line of work presented in this paper, we performed an empirical study evaluating the impact of this embedding in the spreadsheet user's productivity, as we proposed in (Cunha et al, 2012c). The results of this study are presented in (Cunha et al, 2014b) and show that the embedding, together with the evolution mechanism, help users to be more efficient and affective when they work in spreadsheets.

The process of inferring models from existing spreadsheets has been proposed before. Extracting templates from existing spreadsheets was described in (Abraham and Erwig, 2006). That technique is based on similarity of groups of cells and depends on the discovery of repeating patterns in spreadsheets. This technique works very well for spreadsheets that do exhibit such repeating blocks. Our approach is more suitable when relationships exist between data. Another difference is that the target modeling languages are not the same. Nevertheless, a similar study has been performed in both works.

Hermans et al (2010) describe a technique to automatically infer class diagrams for existing spreadsheets matching them with a set of pre-defined spreadsheet patterns. This technique works very well for spreadsheets following the pre-defined patterns. This approach does not consider other type of patterns (like the data dependencies we compute), and the derived model is not as rich as the ClassSheet one.

Isakowitz et al (1995) proposed a methodology to infer a logical layer from existing spreadsheets. This process is not completely automatic and sometimes requires human intervention. The reverse process is also presented: to build a spreadsheet application reusing the logical models inferred. To separate the model from the data is a concern of both Isakowitz and our work.

Cheng and Rival propose an abstract domain to infer types of parts (for example, columns) of a spreadsheet (Cheng and Rival, 2012). 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, and not quality oriented as ours.

ClassSheets and entity-relationship diagrams are closely related. There have been proposed several algorithms to infer ER diagrams from databases. We have reused some algorithms proposed by Alhajj in (Alhajj, 2003) to help us construct an intermediate model of a spreadsheet.

A transformation of spreadsheets to the relational setting was proposed by Cunha et al (2009b). Based on functional dependencies a schema is generated and a new spreadsheet based on that model is produced. The model is based only on database techniques and is not well adapted to spreadsheets, in particular, the identification of dependencies ignores the peculiarities of spreadsheets.

9 Conclusions

We have developed a technique and a tool that can automatically infer relational models from spreadsheets. We have introduced a technique to map such models to ClassSheets, which are more suitable for spreadsheet design. These models can be of great help for the maintenance of spreadsheets since knowledge about the underlying model can, for example, prevent erroneous operations.

We have adapted and extended a method to infer entity-relationship models from relational databases to work with spreadsheets. The exploitation of layout information that is specific to spreadsheets was instrumental in the successful working of the tool.

An evaluation of our tool on real-world spreadsheets showed encouraging results and demonstrated that the approach is viable and should be pursued further in future work.

Moreover, a study comparing the automatically generated relational models to the relational models produced by a group of database experts was performed and showed that the models generated are comparable in quality to the ones produced by the experts.

References

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

Abraham R, Erwig M, Kollmansberger S, Seifert E (2005) Visual specifications of correct spreadsheets. In: VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE Computer Society, pp 189–196

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

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

Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387

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

Cunha J (2011) Model-based spreadsheet engineering. PhD thesis, Universidade do Minho

Cunha J, Saraiva J, Visser J (2009a) Discovery-based edit assistance for spreadsheets. In: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, pp 233–237

Cunha J, Saraiva J, Visser J (2009b) From spreadsheets to relational databases and back. In: Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation, ACM, New York, NY, USA, PEPM '09, pp 179–188

Cunha J, Erwig M, Saraiva J (2010) Automatically inferring ClassSheet models from spreadsheets. In: Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE Computer Society, VLHCC '10, pp 93–100

Cunha J, Mendes J, Fernandes JP, Saraiva J (2011a) Embedding and evolution of spreadsheet models in spreadsheet systems. In: VL/HCC'11: IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE Computer Society, pp 186–201

Cunha J, Visser J, Alves T, Saraiva J (2011b) Type-safe evolution of spreadsheets. In: Giannakopoulou D, Orejas F (eds) Fundamental Approaches to Software Engineering, Springer, Lecture Notes in Computer Science, vol 6603, pp 186–201

Cunha J, Fernandes JP, Mendes J, Pacheco H, Saraiva J (2012a) Bidirectional transformation of model-driven spreadsheets. In: Hu Z, de Lara J (eds) Theory and Practice of Model Transformations, Springer, Lecture Notes in Computer Science, vol 7307, pp 105–120

Cunha J, Fernandes JP, Mendes J, Saraiva J (2012b) MDSheet: A Framework for Model-driven Spreadsheet Engineering. In: Proceedings of the 34rd International Conference on Software Engineering, ACM, ICSE'12, pp 1412–1415

Cunha J, Fernandes JP, Mendes J, Saraiva J (2012c) Towards an evaluation of bidirectional model-driven spreadsheets. In: User evaluation for Software Engineering Researchers, ACM Digital Library, USER' 12, pp 25–28

Cunha J, Fernandes JP, Saraiva J (2012d) From Relational ClassSheets to UML+OCL. In: the Software Engineering Track at the 27th Annual ACM Symposium On Applied Computing (SAC 2012), Riva del Garda (Trento), Italy, ACM, pp 1151–1158

Cunha J, Mendes J, Saraiva J, Visser J (2014a) Model-based programming environments for spreadsheets. Science of Computer Programming (SCP) DOI http://dx.doi.org/10.1016/j.scico.2014.02.002

Cunha J, ao P Fernandes J, Mendes J, Saraiva J (2014b) Embedding, evolution, and validation of spreadsheet models in spreadsheet systems. Tech. Rep. TR-HASLab:01:2014, HASLab - High-Assurance Software Laboratory & Universidade do Minho, URL `http://alfa.di.uminho.pt/~jacome/down/tr_embedding.pdf`

Engels G, Erwig M (2005) ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications. In: Redmiles D, Ellman T, Zisman A (eds) 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, USA, ACM, pp 124–133

Erwig M (2009) Software Engineering for Spreadsheets. IEEE Software 29(5):25–30

EuSpRIG (2010) European Spreadsheet Risks Interest Group, `http://www.eusprig.org/`

Hermans F, Pinzger M, van Deursen A (2010) Automatically extracting class diagrams from spreadsheets. In: Proceedings of the 24th European conference on Object-oriented programming, Springer-Verlag, Berlin, Heidelberg, ECOOP'10, pp 52–75

Höst M, Regnell B, Wohlin C (2000) Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. Empirical Software Engineering 5(3):201–214, DOI 10.1023/A:1026586415054, URL `http://dx.doi.org/10.1023/A:1026586415054`

Isakowitz T, Schocken S, Lucas HC Jr (1995) Toward a logical/physical theory of spreadsheet modeling. ACM Trans Inf Syst 13(1):1–37

Lämmel R, Visser J (2003) A Strafunski Application Letter. In: Dahl V, Wadler P (eds) Proc. of Practical Aspects of Declarative Programming (PADL'03), Springer, LNCS, vol 2562, pp 357–375

Maier D (1983) The Theory of Relational Databases. Computer Science Press

Novelli N, Cicchetti R (2001) Fun: An efficient algorithm for mining functional and embedded dependencies. In: ICDT '01: Proceedings of the 8th International Conference on Database Theory, Springer-Verlag, London, UK, pp 189–203

Panko R (2000) Spreadsheet errors: What we know. what we think we can do. In: Proceedings of the European Spreadsheet Risks Interest Group Annual Conference (EuSpRIG), pp 7–17

Perry DE, Porter AA, Votta LG (2000) 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, pp 345–355

Powell SG, Baker KR (2003) The Art of Modeling with Spreadsheets. John Wiley & Sons, Inc., New York, NY, USA

Rajalingham K, Chadwick D, Knight B (2001) Classification of spreadsheet errors. In: Proceedings of the European Spreadsheet Risks Interest Group Annual Conference (EuSpRIG), pp 23–34

Ullman JD, Widom J (1997) A First Course in Database Systems. Prentice Hall

Visser E (2005) A survey of strategies in rule-based program transformation systems. Journal of Symbolic Computation 40:831–873

Visser J, Saraiva J (2004) Tutorial on strategic programming across programming paradigms. In: 8th Brazilian Symposium on Programming Languages, Niteroi, Brazil