Complexity Metrics for ClassSheet Models *

Jácome Cunha[1,2], João Paulo Fernandes[1,3], Jorge Mendes[1], and João Saraiva[1]

[1] High-Assurance Software Laboratory (HASLab/INESC TEC) &
Universidade do Minho, Portugal
{jacome,jpaulo,jorgemendes,jas}@di.uminho.pt
[2] CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal
jmc@estgf.ipp.pt
[3] Reliable and Secure Computation Group ((rel)ease),
Universidade da Beira Interior, Portugal
jpf@di.ubi.pt

Abstract. This paper proposes a set of metrics for the assessment of the complexity of models defining the business logic of spreadsheets. This set can be considered the first step in the direction of building a quality standard for spreadsheet models, that is still to be defined.
The computation of concrete metric values has further been integrated under a well-established model-driven spreadsheet development environment, providing a framework for the analysis of spreadsheet models under spreadsheets themselves.

Keywords: Spreadsheets, Models, ClassSheets, Metrics, Quality

1 Introduction

Spreadsheet systems are paradigmatic in terms of widespread use and success. Indeed, spreadsheets are intensively used in industry in the development of business applications specially by non-professional programmers, often referred to as *end users*.

The reasons for the tremendous commercial success that spreadsheets experience undergoes continuous debate, but it is almost unanimous that two key aspects deserve to be recognized: i) spreadsheets are highly flexible, which inherently guarantees that they are intensively multi-purpose; ii) the initial learning effort associated with the use of spreadsheets is objectively low.

It is also widely accepted that spreadsheets, in contrast with their success, tend to be highly error-prone. In this line, several studies can be referenced [14,

* This work is part 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 projects FCOMP-01-0124-FEDER-010048, FCOMP-01-0124-FEDER-020484, and FCOMP-01-0124-FEDER-022701. The three first authors were funded by FCT grants SFRH/BPD/73358/2010, SFRH/BPD/46987/2008, and BI2-2012_PTDC/EIA-CCO/108613/2008_UMINHO, respectively.

16, 17], including one showing that up to 94% of all spreadsheets contain errors [13]. Also, there is a long and frequently-updated list of horror stories that directly involve spreadsheets, which is maintained by the *European Spreadsheet Risks Interest Group*[4]. It will only take a minute scrolling this list for the interested reader to be acquainted with how easy it is to cause great (mostly financial) damage using a simple spreadsheet.

In an attempt to address some of the issues that arise from the use of spreadsheets, Engels and Erwig [9] proposed the use of models, namely ClassSheets, to abstractly define the business logic of spreadsheet data. The key idea is that it is easier to understand, to maintain, and to develop such abstract an concise business logic models than the corresponding, possibly large and complex, spreadsheet data. Furthermore, they also proposed a first attempt to use Model-Driven Engineering (MDE) in the context of spreadsheets: from the ClassSheet model a first spreadsheet (i.e. instance) is produced. This a *standard* spreadsheet where some of the business logic, expressed in the model, is embedded as spreadsheet formulas and visual objects. Such a generated spreadsheet guides end users inputting correct data, thus avoiding errors. Recently, we have extended this work to provide a full MDE experience [5]: both ClassSheet models and spreadsheet data are defined in a widely used spreadsheet system where the same visual spreadsheet representation and user interaction is provided for both software artifacts. Most importantly, we developed techniques to allow end users to evolve either the model or its instance, having the correlated artifact automatically updated [4]. These techniques are implemented in the MDSheet framework [6].

The approach provided by the MDSheet framework highly resembles the way a civil engineer thinks, for example, of a house: first, a model is defined and thoroughly evolved, and only then a house is actually built. During this evolution process the engineer computes metrics to reason about the model/house: for example, determining its area, the number of stairs needed between floors, etc. Furthermore, he/she also needs to reason/understand the complexity and quality of the model so that the construction of the house does not become impossible or too expensive. When developing a ClassSheet model, or when evolving an existing one, we face the same problem: we need to reason about the model so that we evolve it in the right direction. That is to say that just like civil engineers, we need metrics for ClassSheet models so we can understand their complexity and quality. However, no such set of metrics has been proposed so far.

In this paper, we build on this limitation to make a first step towards the construction of the first quality standard for spreadsheet models. Indeed, we propose a comprehensive and representative set of metrics that can be used to provide complexity considerations of such models, being this the first major contribution of this paper. The second major contribution of the paper is the empirical analysis of the proposed metrics. Indeed, we apply our metrics to a significant set of spreadsheet models, and we suggest how the value of each metric may

[4] The horror stories are available at `http://www.eusprig.org/horror-stories.htm`.

positively or negatively influence the quality of the overall model. Finally, the third contribution of the paper is that we have implemented the computation of the metrics that we propose under the environment of [6]. This means that for any spreadsheet model under such environment, users can automatically obtain its corresponding value for each of the proposed metrics.

The remaining of this paper is organized as follows. In Section 2 we revise the spreadsheet modeling framework under which we propose to analyze the complexity of spreadsheet models. In Section 3 we introduce in detail the set of metrics that we propose to use in our analysis, and in Section 4 we describe its implementation. Finally, in Section 5 we compare our work with related works and in Section 6 we conclude the paper and point some directions for future research.

2 Modeling Spreadsheets with ClassSheets

In this paper, we propose a set of metrics for spreadsheet models. In particular, we focus our attention on a particular type of well-established and well-studied modeling framework for spreadsheets: the ClassSheet framework [9].

ClassSheets are a high-level, object-oriented formalism to specify the business logic of spreadsheets. ClassSheets allow users to express business object structures within a spreadsheet using concepts from the Unified Modeling Language (UML) such as *classes* and *attributes*, and in fact a mapping from ClassSheets to UML has already been proposed [8]. Using ClassSheet models, it is possible to define spreadsheet tables and to give them names, to define labels for the table's columns, to specify the types of the values such columns may contain and also the way the table expands (e.g. horizontally or vertically).

Using ClassSheets, spreadsheet development is triggered by the definition of a ClassSheet model, that abstractly defines the structure of a spreadsheet, from which a concrete spreadsheet instance, in which actual data is to be inputted, is derived. In order to achieve a practical spreadsheet development environment, we have in the past proposed to embed ClassSheet models in spreadsheets themselves [5]. This feature was further fully integrated in a widely used spreadsheet system [6], and our approach provided the first coherent and single environment for creating and evolving spreadsheet models while automatically obtaining conforming instances: the ClassSheet model is defined in one worksheet of the spreadsheet while the conforming data is updated in another worksheet of that same spreadsheet.

In Figures 1 and 2 we show an example that was built under the environment that we have developed: in Figure 1 a ClassSheet model for a personal budget spreadsheet was defined, and in Figure 2 we can already see that some concrete values for, e.g. incomes such as salary and expenses with housing were already inserted in the spreadsheet that one obtains from the model defined previously. Actually, incomes and housing expenses are ClassSheet **classes** that are **vertically expansible** (indicated in the model by rows 6 and 12 being filled with

ellipsis): this means that in the data instances as many concrete incomes and housing expenses as necessary are possible. One may also note that the model foresees multiple years for a budget, with year being an **horizontally expansible class** (indicated in the model by column `I` being filled with ellipsis). Also, ClassSheet models are attribute-based: months, for example, are **attributes** of the year class. This is indicated with lower-case names such as `jan` or `feb` in the model, but derived attributes are also present, like `profitsjan` or `balancejan`. These derived attibutes consist in spreadsheet formulas, where a standard formula is used as the attribute value, and references are attribute names instead of usual cell references. Moreover, ClassSheets can also have **relationships**: the attributes that intersect both a class that expands vertically and a class that expands horizontally form a relationship. This is the case of the cells `B5` to `H5` which are the attributes of the relationship between the class profits (that expands vertically) and the class year (that expands horizontally).

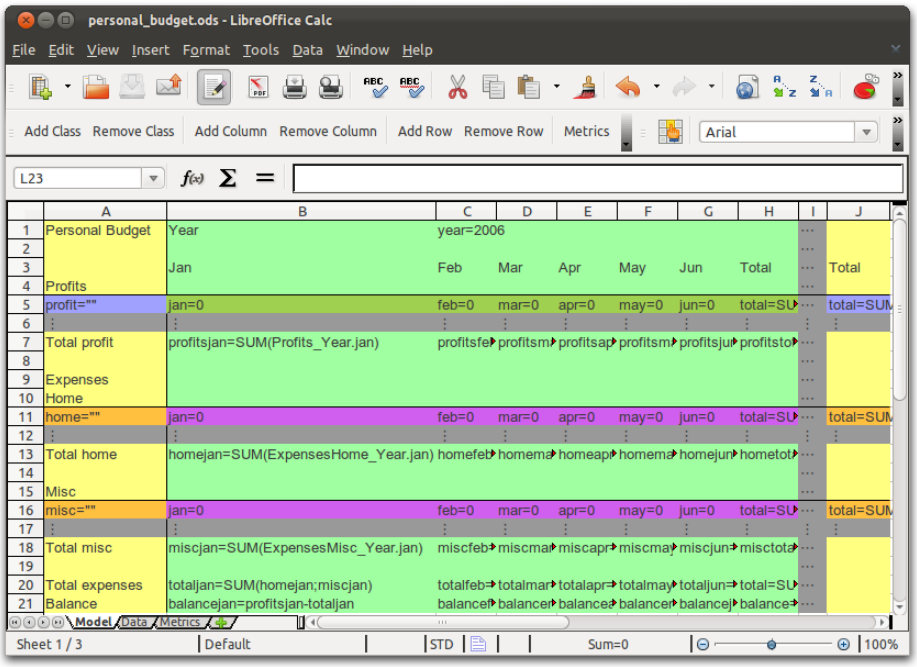

Fig. 1: An abstract model for a personal budget spreadsheet.

In the remaining of this paper, we will use the budget spreadsheet model and its instance as a running example. Also, in our current work we follow the same philosophy that we have followed in the past: we have chosen to integrate the calculation of the metrics and the visualization of its results under a traditional spreadsheet environment. Indeed, one may already observe from Figures 1 and 2

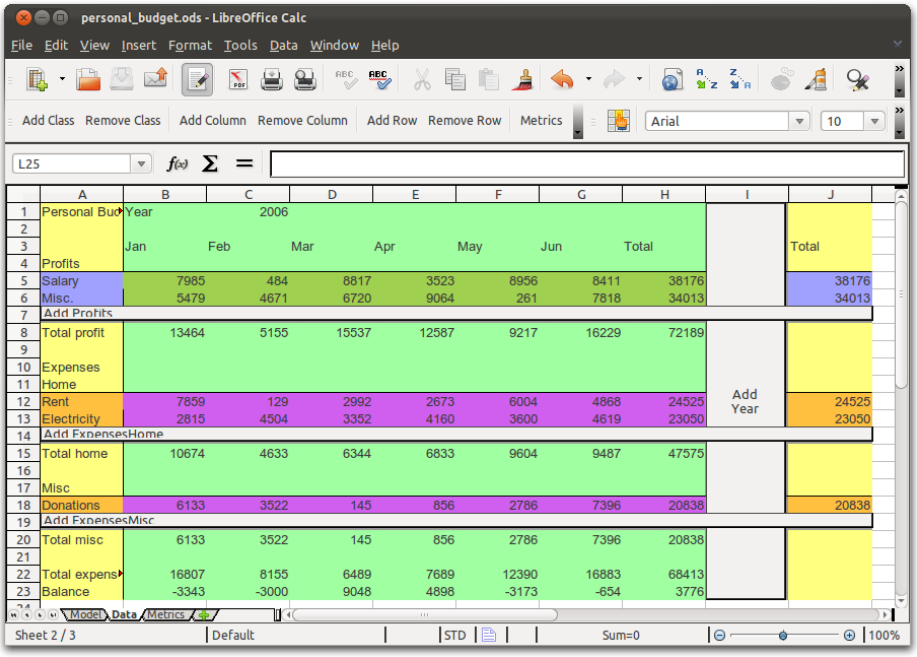

Fig. 2: A concrete spreadsheet for a person's budget for the first semester of a year.

that a third worksheet named Metrics has been added to our running environment, and it is in this worksheet that the results of the metrics that we propose are presented (both numerically as well as graphically), in a way that we describe in detail in Section 4.

3 Metrics for Spreadsheet Models

In [10] the authors describe a set of metrics to analyze the complexity of Entity Relationship (ER) diagrams. These metrics are easy to understand and easy to be interpreted. Unfortunately, they were designed to work only with ER diagrams, and not in the context of spreadsheets.

In this section we will explain how the metrics presented in [10] can be adapted for spreadsheet models. More precisely, we will describe in detail how to adapt each metric to work with ClassSheets.

3.1 Why Entity Relationship Metrics Work for ClassSheets

As we described in Section 2, ClassSheets are a high level formalism with concepts like *classes* and *attributes*. This high level concepts allow to compare ClassSheet models with other paradigms, like, for example, entity relationship diagrams. In fact, in previous work we have shown that it is possible to infer

ClassSheet models from existing spreadsheets [3]. This inference technique has several steps, including the creation of an intermediate entity relationship diagram to represent the business logic of the spreadsheet data. It is based on this diagram that we can compute a ClassSheet representing the spreadsheet under consideration.

Thus, there is a close connection between entity relationship diagrams and ClassSheet models, which can be explored in the context of metrics. In the flowing sections we will show that the metrics previously presented for ER diagram can be adapted to work on ClassSheets. We will explain in detail how to perform this task.

3.2 Relationships–Classes Ratio Metric

This metric was originally designed to measure the relation between the *number of relationships* and the *number of entities* in an ER diagram.

From the work we presented in [3], we know that an entity in an ER diagram is represented by a *class* in a ClassSheet. In fact, a ClassSheet class has a similar meaning to an ER entity since both represent some kind of real world entity. In the spreadsheet example we presented in the previous section, the home expenses is a class. Another class is the year. Thus, the number of ER entities is calculated in ClassSheets as the *number of classes*. Next, we show the function that calculates this part of the metric. Note that we use Haskell [15] notation to express our functions.

```
nC = subtract 1 . length . filter (not . classExpandsB) . classes
```

This implementation is very simple: we get the classes from the model, filter out some special classes that do not represent entities, and subtract one unit (there is a class involving the model that should not be counted).

The second part of the metric, the number of relationships, is calculated in a similar way to the ER diagrams. In ClassSheets it is possible to have one attribute of one class referencing another attribute (similar to a foreign key in the database realm). Also, there are some special classes called *cell classes* which represent $M : N$ relationships. For our running example, the cells `B5:H5` in the ClassSheet model (Figure 1) compose a cell class, that is, a relationship. Moreover, formulas with references also represent a kind of relationship. For instance, in the running example, the cell `B20` references the two expenses classes, namely Home and Misc. Thus, representing a relationship from the class personal budget to the expenses classes.

So, to compute the *number of relationships*, we calculate the number of attributes that reference other classes, inside or outside formulas, plus the number of cell classes. If there is more than one attribute in one class referencing another class, this counts as several relationships, since each reference adds some complexity to the model.

Next, we show the function that calculates the number of relationships:

```
nR m = fromTabs + fromOthers
  where
    fromTabs = sum $ map ((+2) . length . nub . g m) cbs
    fromOthers = sum $ map (length . nub . f m) cs
    cbs = filter classExpandsB $ classes m
    cs = filter (\c -> not $ or $ map (classIntersects c) cbs)
      $ filter (uncurry (||) . (classExpandsH /\ classExpandsV))
      $ classes m
```

This implementation is more complex than the last one and we refer to the documentation of the tool for a better explanation of such function. In fact, we will not show the implementation when its complexity makes the comprehension of the work decrease, like in this case.

Finally, the metric is calculated using the following formula (all the formulas are the same as proposed in [10]):

$$RvsC = \left(\frac{nR}{nR + nC}\right)^2$$

The computed value for this metric, and for all the others we will present next, is always bounded by the interval $[0, 1]$. This helps to visualize and interpret the metric results [10].

Intuitively, the greater the number of relationships, the greater the complexity, specially if there are few classes.

For the example ClassSheet presented in Figure 1, this metric is $RvsC = (3/(3+4))^2 = 0.184$.

3.3 Class Attributes–Classes Ratio Metric

The second metric we adapt to ClassSheets was introduced in [10] to calculate the relation between the *number of entity attributes* and the *number of entities* in an ER diagram.

As we explained in the previous metric, we know that an ER entity is represented in ClassSheets by a class. Thus, the attributes of an entity are the attributes of a ClassSheet class. For instance, in the running example `jan` and `fev` are attributes of the class home expenses. Thus, the metric transposes well to the ClassSheet realm.

The attribute count is done using the following function:

```
nCA =
 length . filter (cellIsFormula) . concat . grid_to_lists . grid
```

The function simply calculates the number of cells that are formulas, which, in this context, means the cells that are attributes.

The function to count of entities was already shown in the previous metric. Next, we show the formula to compute this metric:

$$CAvsC = \left(\frac{nCA}{nCA + nC}\right)^2$$

Intuitively, the greater the number of attributes in a class, the greater its complexity is.

For our running example, the metric evaluates as $CAvsC = (63/(63+4))^2 = 0.915$.

3.4 Relationship Attributes–Relationships Ratio Metric

This metric was originally designed to measure the relation existing between the *number of relationship attributes* and the *number of relationships* in an ER diagram.

As in entity relationship diagrams, ClassSheets can also have relationships with attributes. For instance, in the running ClassSheet example, cell `B5` (with the content `jan=0`) is an attribute of the relationship between year and income. In fact, all the cells in the range `B5:H5` are attributes of such relationship.

To compute the number of relationship attributes we use the following function:

```
nRA = length . filter (isRel) . concat . grid_to_lists . grid
  where isRel (CellFormula (Formula _ (ExpRef _ _ ))) = True
        isRel _ = False
```

The function gathers all the cell classes, that we previously identified as being relationships, and counts all the attributes composing them.

The function to compute the number of relationships was already identified. The formula to compute this metric is given next:

$$RAvsR = \left(\frac{nRA}{nRA + nR}\right)^2$$

The output of this metric means, intuitively, that the greater the number of attributes, the greater the relationship complexity.

For our running example, this metric evaluates to $RAvsR = (88/(88+3))^2 = 0.935$.

3.5 M:N Relationships–Relationships Ratio Metric

The metric we now explain was first introduced to measure the *number of M:N relationships* compared with the *total number of relationships* in an ER diagram.

In our realm, M:N relationships also exist. In the running example, the relationship between year and income is of this kind. This happens because for each year the spreadsheet can have several lines of income, and each kind of income can appear in several years. Thus, the metric can be adapted to work on ClassSheets.

To calculate the *number of M:N relationships*, we designed the function `nMNR`. Since it is a complex function, and since it would not help the reader to better understand our work, we do not show it here and refer to our implementation for more details.

The function to calculate the number of relationships was already described. To compute the complete metric, we use the following formula, again as in [10]:

$$MNRvsR = \frac{nMNR}{nR}$$

The higher the number of M:N relationships, the higher the metric will measure. In the case of ClassSheet models, and also in ER diagrams, M:N relationships are more complex to handle than other relationships.

In the running example, this evaluates to $MNRvsR = 3/3 = 1$.

3.6 1:N and 1:1 Relationships–Relationships Ratio Metric

Analogously to the previous metric, this one measures the relation between the *number of 1:N and 1:1 relationships* and the *total number of relationships*.

We have implemented in Haskell the function `n1N&11`, but as in the previous metric, given its complexity, we refer to the implementation for further details. Nevertheless, if a relationship is not of the kind M:N, then it must be of 1:N or 1:1 kind.

Finally, the ratio is calculated calculated using the formula:

$$1N\&11vsR = \frac{n1N\&11}{nR}$$

Intuitively, it is better to have more 1:N and 1:1 relationships than M:N. Thus, the higher this measure gets, the best.

Unfortunately, our running example does not have 1:N nor 1:1 relationships, and thus the metric measures 0.

3.7 N-ary Relationships–Relationships Ratio Metric

It is common to have n-ary relationships in ER diagrams. Thus, this metric was introduced to measure the relation between the *number of n-ary relationships* and the *total number of relationships*.

As in the ER realm, in ClassSheets it is possible to have n-ary relationships. This can be computed counting the number of references outgoing from a binary relationship, which are given by cell classes, as explained in the next metric. To do this, it is necessary to add the number of classes that reference other classes and are referenced back by the same class.

To compute the number of n-ary relationships, we have implemented the function `nNaryR`:

```
nNaryR m = (length . filter ((>0). length . nub . g m) .
              filter classExpandsB . classes) m
```

Essentially, this function computes the number of cell classes, plus the number of classes that reference other classes and are referenced back by the same class.

The metric is computed calculating the ratio between the number of n-ary relationships and the total number of relationships:

$$NaryRvsR = \frac{nNaryR}{nR}$$

Similarly to M:N and 1:N relationships, n-ary relationships are more complex than binary relationships. Thus, intuitively, it is preferable to have this measure as low as possible.

Unfortunately, our running example does not have n-ary relationships, and thus the metric measures 0.

3.8 Binary Relationships–Relationships Ratio Metric

The last metric we adapt computes the relation between the *number of binary relationships* and the *total number of relationships*.

As ClassSheet models can have n-ary relationships, they can also have binary ones. In fact, if a relationship is not n-ary, it is binary. Thus, the function computes the total number of relationships minus the number of n-ary ones.

The metric is thus calculated through the formula:

$$BRvsR = \frac{nBR}{nR}$$

Once more, it is preferable to have this measure higher, meaning that most relationships in the model are binary and not n-ary.

For our running example, $BRvsR = 3/3 = 1$.

3.9 Formulas–Cells Ratio Metric

All the metrics we described until now are adapted from entity relationship diagrams and are quite interesting to give complexity related to entities/classes, relationships, and their attributes. But this is not enough for spreadsheet models. One of the main issues related to spreadsheets, and their models, is formulas and their complexity. Thus, we need to introduce new metrics that give some measures about them.

The first metric we introduce computes the relation between the *number of formulas* and the *number of cells*.

The number of formulas in a ClassSheet is given by the following function:

```
nF m = length $ catMaybes $ concat $ grid_map (aux) (grid m)
  where
    aux _ (CellFormula (Formula _ (ExpFun _ _))) = Just ()
    aux _ (CellFormula (Formula _ (ExpBinOp _ _ _))) = Just ()
    aux _ (CellFormula (Formula _ (ExpPar _))) = Just ()
    aux _ _ = Nothing
```

This functions traverses a ClassSheet model and returns the size of the list of formulas.

The function that calculates the number of cells is shown next, and counts all the cells in the model that are not empty:

```
nCe = length . filter (/= (CellValue $ VText "")) .
         concat . grid_to_lists . grid
```

The final metric if given by the formula:

$$FvsCe = \frac{nF}{nCe}$$

Intuitively, the greater the number of formulas in a model, the greater the complexity of such model.

For the running example, this metric computes the value $FvsCe = 41/130 = 0.315$.

3.10 Formula References–Formulas Ratio Metric

As the previous metric, this one reports to formulas, in particular, it gives the relation between the *number of references in formulas* and the *number of formulas*. Note that all the other references in the model are already used to calculate other metrics. In fact, the formula references were the only ones not being explored.

The function, `nRe`, that computes the references of a formula is complex, and thus we do not show it here.

The function that calculates the number of formulas in a ClassSheet model was described in the previous metric. The formula that computes this metric is now given:

$$RevsF = \left(\frac{nRe}{nRe + nF}\right)^2$$

Intuitively, the greater this measure is the more complex the model we have, that is, if each formula has many references, it will be more difficult to handle.

For our running example, the measure computed is $RevsF = (88/88 + 41)^2 = 0.465$.

3.11 Size Metrics

This last metric is also not adapted from [10]. In fact, it calculates a set of general measures about a model, namely (in parenthesis we show the metrics computed for our running example),

- the total number of non-empty cells (130);
- the width of the model, that is, the number of columns (10);
- the height, that is, the number of rows (21);
- the number of expandable classes (4);

– the number of input cells (22);
– the number of output cells (41).

These metrics are implemented traversing the ClassSheet model and counting the corresponding characteristic, which can be seen in the complete implementation.

In the next section we will show in detail the tool we have designed and implemented to make these metrics available to users in the environment they are used to, that is, in a spreadsheet system.

4 The MDSheet Framework

In the course of our work, we developed an OpenOffice/LibreOffice extension named MDSheet [6]. This tool was first created to implement the embedding of ClassSheet models, with the possibility to evolve them having the data automatically coevolved [5]. Then, we further improved it with bidirectional transformations allowing users to evolve the data without being concerned with the model since the tool synchronizes them automatically [4]. The metrics described in the previous section were also implemented in the MDSheet framework, being the implementation explained in this section.

In the user interface, a new button is available in the toolbar (see Figure 3) to evaluate the metrics for the current model. When such button is pressed, a new

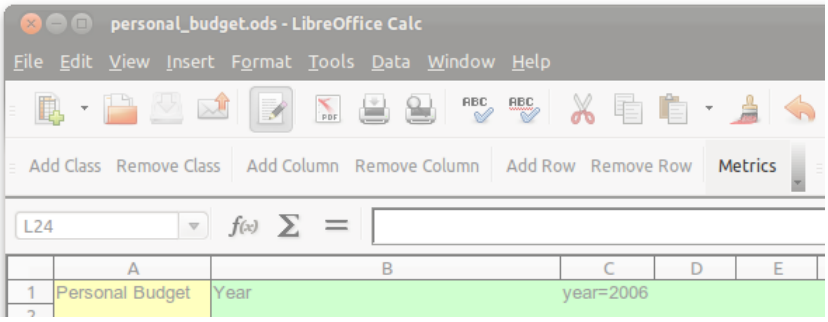

Fig. 3: MDSheet toolbar with button to evaluate the metrics of the current model.

worksheet, named `Metrics`, is created with the metrics for the current model, but also with the average of the metrics for the models available in our public repository of ClassSheet models. An example of such worksheet is depicted in Figure 4, where column `A` contains the name of the metrics, column `B` contains the results obtained from running the metrics for the current model, and column `C` contains the average of the metrics gathered from the repository.

The goal of comparing the metrics of the current model with metrics from other models is to quickly provide a relative estimate of the quality of the current

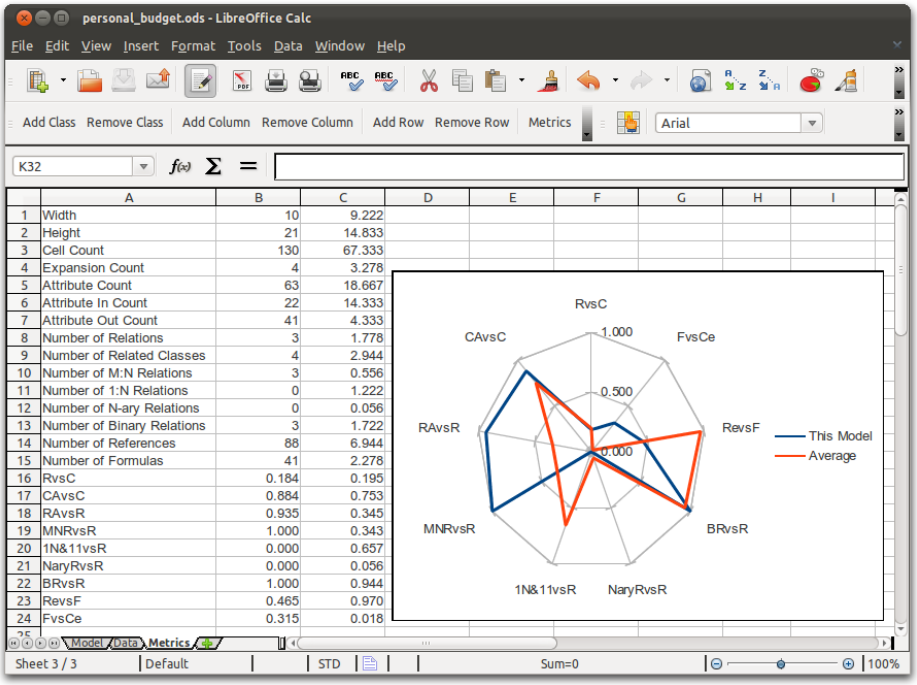

Fig. 4: Sheet with metrics generated by MDSheet for the running example.

model. The repository is still a work in progress, and we aim to develop and collect a comprehensive set of models, such that new users can start working directly based on a model already available, but also that modelers can use as a starting or reference point for their work. When the repository becomes large enough, we will be able to group the models by domains (e.g., finances or database), and users will have the possibility to compare their model with others from the same domain, providing a better estimate of the model quality.

When the models repository is updated, so is the MDSheet framework, so that it contains the most recent metrics.

The MDSheet framework is developed in such a modular way that it can be very easily improved with the addition of new features, such as we did with the metrics evaluation. The framework is divided in three main parts, as can be seen in Figure 5:

Core — This part, developed entirely in Haskell, deals with the abstract representation of the models and data. This allowed to define a bidirectional transformation system at a high level, and also to specify and integrate the metrics for the models. Currently, this part is only composed by the already mentioned transformation system and the metrics evaluation code.

Interface — This part is the one that users interact with. It is mainly developed in OpenOffice Basic (an idiom of the Basic language). For the metrics

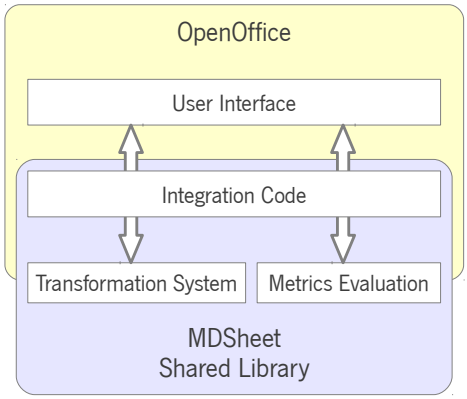

Fig. 5: Architecture of our OpenOffice/LibreOffice environment with the MDSheet extension.

evaluation, it sends a request to the core through the integration code to evaluate the metrics of the model and then creates the new worksheet with the result in it.

Integration Code — This part is used to connect the interface to the core of the MDSheet framework. This connection, developed in C/C++ and Haskell, is entrusted with the invocation of the core functions and also with the conversion of the data from the interface to the core and vice versa.

The MDSheet framework (both source code and compiled version) is available at:

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

5 Related Work

Metrics for spreadsheets have been defined [1, 7, 12] by different authors. Unfortunately, such metrics do not work well for spreadsheet models. In this work, we adapted and extended existing metrics from the modeling realm to spreadsheet models. To the best of our knowledge, this is the first attempt of such a work.

In [2] the authors take a first step towards automated assessment of spreadsheet maintainability. They apply the selected metrics to the EUSES spreadsheet corpus in order to study their behavior. Their work aims to achieve a maintainability model for spreadsheets, whilst we defined a first set of metrics for spreadsheet models. These metrics can now be used to construct, for example, a maintainability model for spreadsheet models.

In [11] the authors sketch a new maintainability model that alleviates some problems reported by other techniques. Since this work was done in an industrial environment, the authors discuss their experiences with using such a model for IT management consultancy activities. Again, their model features maintainability

of software. With our work we intend to make available the tools, i.e., the metrics, necessary to achieve a similar goal, but for ClassSheet models.

6 Conclusion

This paper presents a set of metrics for spreadsheet models, in this case, ClassSheets. We adapted metrics from a different modeling paradigm, namely entity relationship diagrams, because there is an interesting similarity between both modeling fields.

Although all the metrics existing for ER diagrams were adapted, this was not enough. Formulas, which are a concerning issue in spreadsheets, and thus in spreadsheet models, were not considered in ER diagrams metrics, not even for computed values (considering an extended version of ER diagrams). Thus, we introduced some new metrics, in the same line of work of the other metrics, to compute some measures about ClassSheet formulas and their relationship with the underlying model. Moreover, we calculated a set of ClassSheet specific metrics such as the width of the model and the number of cells.

This work represents an interesting basis for constructing a quality standard for ClassSheets. In fact, it could be further developed and used to construct, for example, a maintainability standard or even a complete quality standard for ClassSheets.

References

1. Bregar, A.: Complexity metrics for spreadsheet models. Proceedings of the 2004 European Spreadsheet Risks Interest Group (EuSpRIG) CoRR abs/0802.3895, 85–93 (2004)
2. Correia, J.P., Ferreira, M.A.: Measuring maintainability of spreadsheets in the wild. In: Proceedings of the 27th IEEE International Conference on Software Maintenance. pp. 516–519. ICSM '11, IEEE (2011)
3. Cunha, J., Erwig, M., Saraiva, J.: Automatically inferring classsheet models from spreadsheets. In: Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 93–100. VLHCC '10, IEEE Computer Society (2010)
4. Cunha, J., Fernandes, J.P., Mendes, J., Pacheco, H., Saraiva, J.: Bidirectional transformation of model-driven spreadsheets. In: Hu, Z., de Lara, J. (eds.) Theory and Practice of Model Transformations. Lecture Notes in Computer Science, vol. 7307, pp. 105–120. Springer (2012)
5. Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: Embedding and evolution of spreadsheet models in spreadsheet systems. In: Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 186–201. VLHCC '11, IEEE Computer Society (2011)
6. Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: MDSheet: A Framework for Model-driven Spreadsheet Engineering. In: Proceedings of the 34rd International Conference on Software Engineering. pp. 1412–1415. ICSE '12, ACM (2012)

7. Cunha, J., Fernandes, J.P., Peixoto, C., Saraiva, J.: A quality model for spreadsheets. In: Proceedings of the 8th International Conference on the Quality of Information and Communications Technology, Quality in ICT Evolution Track. pp. 231–236. QUATIC '12 (2012)
8. Cunha, J., Fernandes, J.P., Saraiva, J.: From Relational ClassSheets to UML+OCL. In: Proceedings of the Software Engineering Track at the 27th Annual ACM Symposium On Applied Computing. pp. 1151–1158. SAC '12, ACM (2012)
9. Engels, G., Erwig, M.: ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications. In: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. pp. 124–133. ACM (2005)
10. Genero, M., Jiménez, L., Piattini, M.: Measuring the quality of entity relationship diagrams. In: Proceedings of the 19th International Conference on Conceptual Modeling. pp. 513–526. ER '00, Springer-Verlag (2000)
11. Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: Proceedings of the International Conference on Quality of Information and Communications Technology. pp. 30–39. QUATIC '07, IEEE Computer Society (2007)
12. Hodnigg, K., Mittermeir, R.T.: Metrics-based spreadsheet visualization: Support for focused maintenance. CoRR abs/0809.3009 (2008)
13. Panko, R.: Facing the problem of spreadsheet errors. Decision Line, 37(5) (2006)
14. Panko, R.: Spreadsheet errors: What we know. what we think we can do. Proceedings of the 2000 European Spreadsheet Risks Interest Group (EuSpRIG) (2000)
15. Peyton Jones, S.: Haskell 98: Language and libraries. Journal of Functional Programming 13(1), 1–255 (2003)
16. Powell, S.G., Baker, K.R., Lawson, B.: A critical review of the literature on spreadsheet errors. Decision Support Systems 46(1), 128–138 (2008)
17. Rajalingham, K., Chadwick, D.R., Knight, B.: Classification of spreadsheet errors. In: Proceedings of the 2001 European Spreadsheet Risks Interest Group (EuSpRIG). Amsterdam (2001)