Information Extraction from Legacy Spreadsheet-based Information System

An Experience in the Automotive Context

Domenico Amalfitano[1], Anna Rita Fasolino[1], Porfirio Tramontana[1], Vincenzo De Simone[1], Giancarlo Di Mare[2] and Stefano Scala[2]

[1]*Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, Università Federico II, Via Claudio 21, Naples, Italy*

[2]*Fiat Group Automobiles S.p.A.,Pomigliano Technical Center,Via ex Aeroporto, Pomigliano d'Arco, Italy*

Keywords: Information Extraction, Legacy Systems, Spreadsheets, Automotive.

Abstract: Nevertheless spreadsheets were originally designed for computing purposes and for commercial applications, they are often used in industry to implement Information Systems, thanks to the functionalities offered by integrated scripting languages and ad-hoc frameworks (e.g., Visual Basic for Applications). This technological solution allows the adoption of Rapid Application Development processes for the quickly development of Spreadsheets-based Information Systems, but the resulting systems are quite difficult to be maintained and very difficult to be migrated to other architectures such as Database-oriented Informative Systems or Web applications. In this paper we present an approach for reverse engineering the data model from an Excel spreadsheet-based information system. The approach exploits a set of heuristic rules that are automatically applied in a seven-steps process. The applicability of the process has been shown in an industrial context where it was used to obtain the UML class diagrams representing the conceptual data models of three spreadsheet-based information systems.

1 INTRODUCTION

Spreadsheets are interactive software applications designed for collecting and analyzing data in tabular form. In a spreadsheet, data are organized in worksheets, any of which is represented by a matrix of cells each containing either data or formulas. Modern spreadsheets applications (e.g. Microsoft Excel) are integrated with scripting languages (e.g., Microsoft Visual Basic for Applications). These languages allow the realization of interactive functionalities for the management of the underlying data by adopting Rapid Application Development processes. In these scenarios, spreadsheets have left behind their original role of calculation sheets, becoming instead core business tools. As a recent analysis showed, the usage of spreadsheets is diffused in a relevant community made by many millions of end-user programmers (Scaffidi, 2005).

Although it is possible to build Spreadsheets-based Information Systems very quickly and without high development costs, their quality is often low. The data often present replication problems; data management tends to be error prone due to the lack of native consistency checking or data visibility restriction mechanisms. Therefore, the user dissatisfaction when working with these systems may increase as the size and the complexity of data increases, so business organizations may be compelled to look for more effective solutions.

Migrating a legacy Spreadsheets-based Information System towards new technologies and platforms often provides an effective solution to these problems. In the last years, several processes, techniques and tools to support the migration of legacy software systems have been proposed in the literature (De Lucia, 2008; Bovenzi, 2003; Canfora, 2008).

The first and more critical step in a migration process consists in the reverse engineering of the data model of the information scattered in the cells of different spreadsheets and worksheets. The reverse engineered model will provide a useful abstraction about the analyzed data and can be considered the starting point for planning any reengineering activity.

In (Amalfitano, 2014a) we presented a process for migrating a legacy Spreadsheets-based Information

Amalfitano D., Fasolino A., Tramontana P., De Simone V., Di Mare G. and Scala S..
Information Extraction from Legacy Spreadsheet-based Information System - An Experience in the Automotive Context.
DOI: 10.5220/0005139603890398
In *Proceedings of 3rd International Conference on Data Management Technologies and Applications* (KomIS-2014), pages 389-398
ISBN: 978-989-758-035-2

System to a new Web application. This process was defined in an industrial context where we reengineered a legacy system used in a tool-chain adopted by an automotive company for the development of embedded systems. According to this company, such system was affected by many maintenance and usability issues, which motivated the migration process.

The first step of the migration process was devoted to the reverse engineering of a data model and of the business rules embedded in the spreadsheet system. The successive steps regarded the reengineering and reimplementation of the system according to the target platform. The first step was performed using a set of heuristic rules that we described in (Amalfitano, 2014b).

In this paper we present a further refinement of our work: we propose a structured reverse engineering process that can be used to analyze spreadsheet-based information systems and to infer a UML class diagram from them. The class diagram provides a conceptual model of the data embedded in the spreadsheets and is comprehensive of classes, class names, class attributes, association and composition relationships. The process is based on the execution of seven sequential steps that can be automatically performed.

The effectiveness of the proposed process has been assessed by a wider experimentation involving more spreadsheets-based information systems from the same industrial domain. In this paper we present the proposed process and the results of the experiment we performed.

The paper is organized as it follows: Section 2 presents related works, while Section 3 illustrates the proposed data model reverse engineering process. Section 4 shows an example of using the process. Section 5 illustrates the experiment we performed to evaluate the approach, while Section 6 presents some conclusive remarks.

2 RELATED WORKS

Due to the wide diffusion of spreadsheets in business and in industry, a great interest in spreadsheet analysis has been recently recorded. Several authors addressed this problem and proposed techniques and tools supporting the automatic analyses of spreadsheets.

Several works in the literature aimed at inferring a data model from spreadsheets. Some of them are based on explicit extraction and transformation rules that need to be provided by the users, such as the technique proposed by Hung et al. (Hung, 2011).

The works of Abraham, Erwig et al. (Abraham, 2004; Abraham, 2006; Abraham, 2007; Abraham, 2009) exploit the existence of the shared template underlying a given spreadsheet corpus, and propose techniques for automatically inferring the templates. The inferred template is hence exploited for safe editing of the spreadsheets by end-users, avoiding possible errors.

Two other approaches with similar purposes are the ones of Mittermeir and Clermont (Mittermeir, 2002) and Ahmad et al. (Ahmad, 2003) that exploit specific information about the type of data contained in the spreadsheet cells.

Cunha et al. (Cunha, 2010), on the basis of the previous results of Abraham and Erwig, propose reverse engineering techniques to derive ClassSheet models from existing spreadsheets by using data mining techniques and infer functional dependencies among columns.

Recently, Chen et al. (Chen, 2013) propose automatic rules to infer some information useful in the migration of a spreadsheet into a relational database. They evaluated the effectiveness of the proposed rules on a very large set including more than 400 thousands of spreadsheets crawled from the Web.

The work by Hermans et al. (Hermans, 2010) addresses a problem very similar to ours that is the recovery of a conceptual data model from spreadsheets. That paper indeed presents a technique for reverse engineering a data model from spreadsheets that is based on two-dimensional patterns. These patterns regard layout, data and formulas included in the spreadsheets and can be specified, recognized and transformed into class diagrams. Some patterns were proposed by the authors, as well as other ones were found in the literature (e.g. Janvrin, 2000; Panko, 1994; Ronen, 1989). The technique has been validated with respect to the spreadsheets corpus proposed by Fisher and Rothermel (Fisher, 2005). Successively, Hermans et al. (Hermans, 2011) evaluated the usefulness of this approach by involving end users from industry and abstracting leveled dataflow diagrams, too.

However, these techniques are mainly applicable to spreadsheets that are used as calculation sheets, and that perfectly follow the predefined patterns. On the contrary these techniques may not be effective for analyzing spreadsheets not conforming to that patterns or used as information systems. In the latter case, indeed, no formula is present in the sheets and

the rules used by the authors to classify cells may be misleading.

The spreadsheets developed in the considered industrial context were used as information systems too and were not calculation sheets. As a consequence, we were not able to reuse the techniques found in the literature as-are, but we had to adapt them to the specific context. In particular, we had to look for heuristic rules applicable to the considered types of spreadsheets and we had to define a process made of a sequence of steps for applying the selected rules. The process we defined will be illustrated in the following section.

3 THE CONCEPTUAL DATA MODEL REVERSE ENGINEERING PROCESS

The industrial context of our work included a large number of spreadsheets, implemented by Excel files, used in the development process of Electronic Control Units (ECU) in the automotive company. They supported the Verification & Validation activities (V&V) and the management of Key Performance Indicators (KPI) about the development process. The spreadsheets inherited from a same template and included some VBA functionalities providing data entry assistance. Moreover, their cells followed well-defined formatting rules (i.e., font, colors, and size of cells) that improved the readability and usability of the spreadsheets, and complied to the following layout rule: all the data concerning the same topic in a single spreadsheet file were grouped together in rows or columns separated by empty cells or according to specific spreadsheets patterns (Hermans, 2010).

Taking the cue from other works proposed in the literature (Cunha, 2010; Hermans, 2010; Abraham, 2006), we founded our data model reverse engineering process on a set of heuristic rules. We defined a process made of seven steps that can be automatically performed in order to infer, with gradual refinements, the UML class diagram of the considered information system. In each step, one or more heuristic rules are executed. Each rule is based on the analysis of one or more spreadsheets belonging to the corpus of spreadsheet files composing the subject information system. In the following we describe the steps of the process.

Step 1: in this step we preliminarily apply *Rule 1* that abstracts a class named Sp whose instances are the Excel files that comply to a same template.

Step 2: in this step we exploit the *Rule 2* that is executed on a single spreadsheet file of the corpus. The rule associates each non empty sheet of the Excel file with a class S_i, having the same name of the corresponding sheet. Moreover, an UML composition relationship between the *Sp* class and each S_i belonging to the file is inferred. The multiplicity of the association on each S_i side is equal to *1*. Figure 1 shows an example of applying this rule on an example spreadsheet.

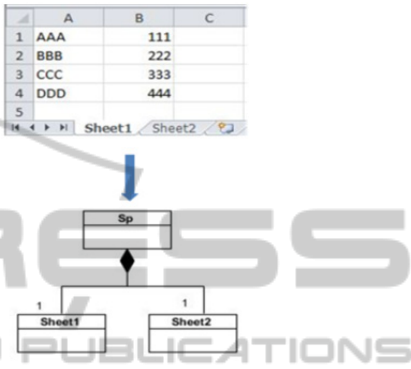

Figure 1: Example of Step 2 execution.

Step 3: in this step we exploit the *Rule 3* that is executed on a single spreadsheet file of the corpus. This heuristic, according to (Abraham, 2006; Hermans, 2010), associates a class A_j for each non empty cell area $Area_j$ of a sheet already associated to a class S_i by the *Rule 2*. Each class A_j is named as follows: *Name of the sheet_Areaj*. Moreover an UML composition relationship between the S_i class and each A_j class is inferred. The multiplicity of the association on each A_j side is equal to *1*. Figure 2 shows an example of Step 3 execution.

Step 4: In this Step two rules *Rule 4.1* and *Rule 4.2* are sequentially executed.

The *Rule 4.1* is applied to discriminate the header cells (Abraham, 2004) of each area $Area_j$ that was inferred in the previous step. *Rule 4.1* analyzes the content of the spreadsheet files belonging to the corpus in order to find the invariant cells for each area $Area_j$. An *invariant cell* of an area is a cell whose formatting and content is the same in all the analyzed spreadsheets.

The set of invariant cells of an area $Area_j$ composes the header of that area. Figure 3 shows how the *Rule 4.1* works. In a first phase (Figure 3-A) all the spreadsheets are queried to select, for each of them, the cells of a given area. In the next phase (Figure 3-B) all the selected cells are analyzed to recognize the invariant cells for the considered area. Intuitively, if we imagine to overlap the contents of

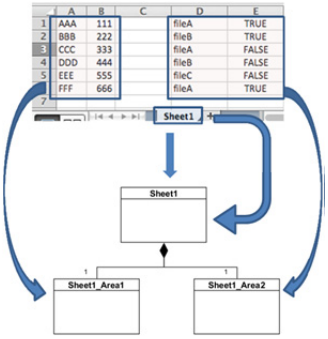

Figure 2: Analysis of non-empty cell areas belonging to Sheet1 executed in Step 3.

of all the spreadsheets for a given area $Area_j$, then the header is given by the cells that, for the considered area, are invariant in all the files, as shown in Figure 3-B.

Rule 4.2 is executed on the headers inferred by *Rule 4.1*. For each area $Area_j$, this heuristic analyzes the style formatting properties of the cells composing its header. It permits to discriminate subareas $SubArea_m$ having header cells satisfying specific patterns. *Rule 4.2* associates a class SA_m for each $SubArea_m$. Each class SA_m is named as follows: *Name of the sheet_SubAream*, if no name could be associated to the class according to the recognized patterns. The names of the attributes of the class are inferred from the values contained into the header cells.

Moreover, an UML composition relationship between the class S_j and each related SA_m class is inferred. The multiplicity of the association on each class SA_m side is equal to *1*. Some examples of *Rule 4.2* executions are reported in Figure 4, 5, 6 and 7.

As an example in Figure 4 a UML class, having the default name *Sheet1_Area1*, is inferred since three consecutive header cells have the same formatting style. The attributes of the class are named after the values contained into the header cells.

Figure 5 shows a variant of the example reported in Figure 4. In this case two UML classes are inferred since two groups of consecutive cells having the same formatting characteristics were found.

A further pattern is shown in Figure 6 where the header is structured in two different levels. The upper level, composed of merged cells, permits to infer a class named *ClassA*, while its attributes are named after the values contained in the header cells of the lower level.

In the example shown in Figure 7, the previous pattern is applied twice and a composition relationship is inferred between the two obtained classes.

Step 5: in this step we exploit *Rule 5* that is applied on the whole corpus of spreadsheets.

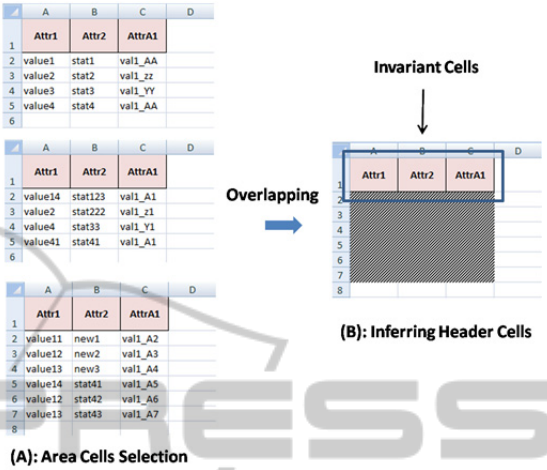

Figure 1: Execution of Step 3.

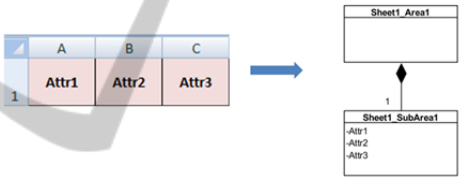

Figure 4: Example of header cells pattern inferring a single class and its attributes.

Rule 5 is applied to all the subareas $SubArea_m$ to find possible sub-subareas made by group of data cells having the same style formatting properties. If a subarea $SubArea_m$ is composed by two or more sub-subareas then the heuristic associates a class, SSA_k for each sub-subarea. Each class SSA_k is named as follows: *Name of the sheet_SubSubAreak*.

Moreover, the new classes substitute the class associated to the $SubArea_m$, and an UML composition relationship between the A_i class and each SSA_k class belonging to the area is inferred. The multiplicity of the association on each class SSA_k side is equal to *1*. Figure 8 shows an example of how Step 5 works.

Step 6: in this step the *Rule 6* is exploited. This heuristic is applied to the overall corpus of spreadsheets. It analyzes the formatting properties of the cells belonging to consecutive subareas and sub-subareas in order to infer association relationships and multiplicities between the classes that were associated to these areas in the previous steps.

An example of how this rule works is reported in Figure 9. In this case two consecutive subareas are considered, i.e., *SubArea1* related to columns A and B and *SubArea2* corresponding to columns C and D. The classes named *Sheet1_SubArea1* and *Sheet1_SubArea2* were associated to *SubArea1* and *SubArea2*, respectively. Since for each spreadsheet a tuple of *SubArea1* is composed by a number of merged cells that is an integer multiple of the number of merged cells related to a tuple of *SubArea2*, then *Rule 6* infers a UML association between the two classes related to the two subareas.

The multiplicity of the association on the class side related to the class *Sheet1_SubArea1* is equal to *1* whereas the one on the other side is *1..**.

Step 7: in this step the *Rule 7* is exploited. This heuristic is applied to the overall corpus of spreadsheets. It analyzes the value of the cells in order to infer association relationships between classes. As an example, if in all the spreadsheets, for each cell of a column/row that was exploited to infer an attribute of a class *A* there is at least a cell of a column/row that was exploited to infer an attribute of a class B having the same value, then it is possible to define a UML relationship between the UML Class A and the UML Class B.

Moreover, if the cells of the column A have unique values then the A side and B side multiplicities of the inferred UML relationship are *1* and *1..** respectively, as shown in Figure 10.

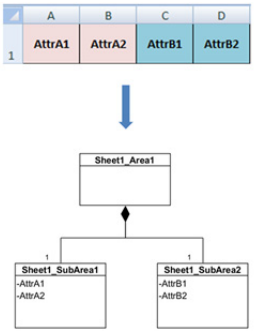

Figure 5: Example of header cells pattern inferring two classes and their attributes.

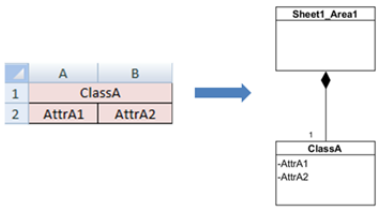

Figure 6: Example of header cells pattern inferring a single class with attributes and class name.

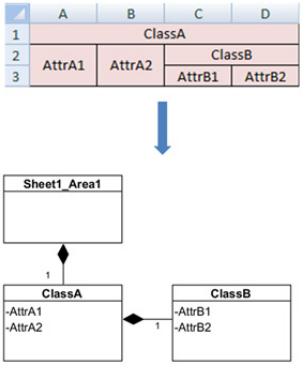

Figure 7: Example of header cells pattern inferring two classes with attributes, class names and their composing relationship.

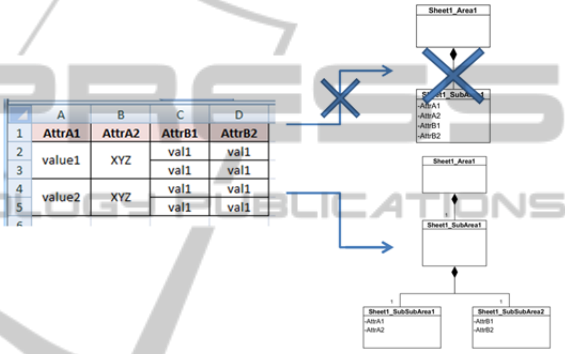

Figure 8: Example of Step 5 Execution.

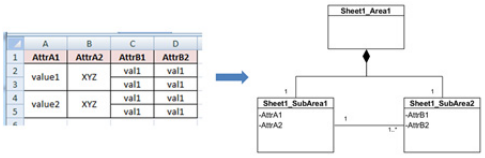

Figure 9: Example of Step 6 Execution.

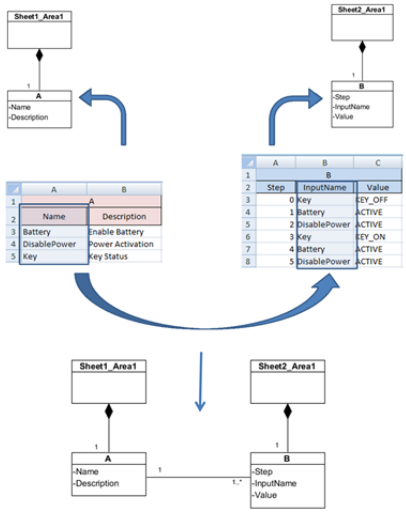

Figure 10: Example of Step 7 Execution.

4 CASE STUDIES

To analyze the effectiveness of the proposed process, we performed a series of case studies involving more spreadsheets-based information systems from the same industrial domain. The goal of these case studies was to evaluate the applicability of the process and the acceptability and precision of the inferred models.

In the first case study we used the proposed process to reverse engineer the conceptual data model from a legacy Spreadsheets-based Information System implemented in Microsoft Excel. This system was used by the HIL (*Hardware-in-the-Loop*) Validation Team of Fiat Group Automobiles and provided strategic support for the definition of high-level testing specifications, named *Test Patterns*. Test Patterns represent essential artifacts in the overall testing process (Shokry, 2009) since they allow the automatic generation of test cases, the *Test Objects*, necessary to exercise the Electronic Control Units (ECUs) of automotive vehicles. The generation process is carried out thanks to a lookup table that embeds both the translation of high-level operations to the corresponding set of low-level instructions, and the mapping between input and output data to the set of ECU pins.

Test Patterns were implemented by means of a predefined Excel template file organized into 7 worksheets, referred to different phases of the testing process.

The worksheets embedded the model of the data. In the considered context, such template has been adopted to instantiate a set of 30,615 different Excel files, constituting the overall spreadsheets-based informative system. These spreadsheets contained on average 2,700 data cells.

All of these files inevitably shared the same structure (i.e., the data model) with a high rate of replicated data (about 20% of data cells recurred more than 1,300 times, while about the 50% more than 100 times). This high rate of replication was mainly due to the fact that data were scattered among multiple spreadsheets. Therefore the underlying model was not normalized, with any information about related data. In addition, the automatic verification of data consistency and correctness was not natively supported.

At the end of the process execution we obtained a UML class diagram composed of 36 classes, 35 composition relationships and 23 association relationships. Seven of these classes are associated with the worksheets composing each spreadsheet file, while the remaining ones are related to the areas and subareas of each worksheet. Figure 11 shows the conceptual data model class diagram that was automatically inferred by executing the process. For readability reasons, we have not reported all the association relationships and the associations' multiplicities, whereas we reported all the classes and the composition relationships that were inferred.

Figure 12 shows an example of rules execution on the Test Patterns sheet that is the most complex one. By executing the rules we were able to infer five classes, 4 association relationships and 5 composition relationships. Moreover, it shows how the Rule 4 were able to infer the two classes Test Step and Expected Results and Rule 6 proposed the association relationship between them.

In the second and third case study, we analyzed two further systems used by the company. The systems included a Software Factory Test Case (called SW TC) repository and a KPI repository (hereafter KPI).

The SW TC repository contains essential artifacts of the overall testing process in the considered company, since they allow the automatic generation of executable MATLAB test scripts necessary to validate the ECU models. This information system is composed by 14,000 Excel files inheriting from a common template composed by *10* sheets. These spreadsheets contained 4,000 data cells on average. About the 75% of data cells are replicated in the spreadsheets.

KPI repository contains the key performance indicators of each development project regarding a software component belonging to a specific type of vehicle. The information system is composed by 1,500 Excel files inheriting from a common template composed by *8* sheets. These spreadsheets contained 1,370 data cells on average. About the 77% of data cells are replicated in the spreadsheets.

Table 1 shows the results of the conceptual data model reverse engineering process involving the three case studies. It shows the number of classes and relationships that were automatically inferred for each information system.

After the analysis, we performed a validation step in order to assess the acceptability and the precision of the models inferred by the reverse engineering process. To this aim we enrolled three experts from the company belonging to the application domains of the information systems.

We submitted them: (1) the inferred data models, (2) a report containing the description of each inferred UML item and one or more traceability

Table 1: Reverse Engineering Results.

Information System	#Classes	#Association Relationships	#Composition Relationships
SW TC	49	33	48
Test Pattern	36	23	35
KPI	25	12	24

links towards the spreadsheet's parts from which it was extracted, (3) a set of questions in order to collect the expert judgments about the validity of the UML item proposals. We asked the experts to answer the questions and thus we were able to assess the effectiveness of the overall reverse engineering process by means of the *Precision* metric reported below:

$$Precision = \frac{|V.I.E.|}{|I.E.|} \times 100$$

I.E. is the set of the UML elements, i.e., classes, class names, class attributes, relationships between classes and multiplicities of the relationships, that were inferred by the process. $|I.E.|$ is the cardinality of the *I.E.* set.

$V.I.E. \subseteq I.E.$ is the number of the inferred UML elements that were validated by the industrial expert and $|V.I.E.|$ is the cardinality of this set.
Table 2 shows the precision values we obtained for each information system.

Table 2: Evaluation Results.

Information System	Precision
SW TC	81%
Test Pattern	84%
KPI	92%

The precision values reported showed that more than 80% of the inferred elements were validated by the experts.

In the remaining cases, the experts proposed some minor changes to the candidate classes. As an example, with respect to the first system, the Test Patterns one, the expert decided to candidate additional classes by extracting their attributes from the ones of another candidate class, or to merge some of the proposed classes into a single one.

In particular, with respect to the class diagram reported in Figure 12, he proposed to (1) discard the *PlotOutput* class and to move its attributes into the *ExpectedResult* one and (2) to extract a new class, named *Repetition*, having the attributes *repetition* and *eventTime* that were given from the candidate class named *Test Step*. Definitely, the expert proposed 1 class split and 2 classes merges operations. Similar changes were proposed also in the other two case studies we performed.

Moreover, we further analyzed the results of the validation steps, in order to assess the applicability of the rules used throughout the process. In this way, we were able to learn some lessons about the heuristic rules.

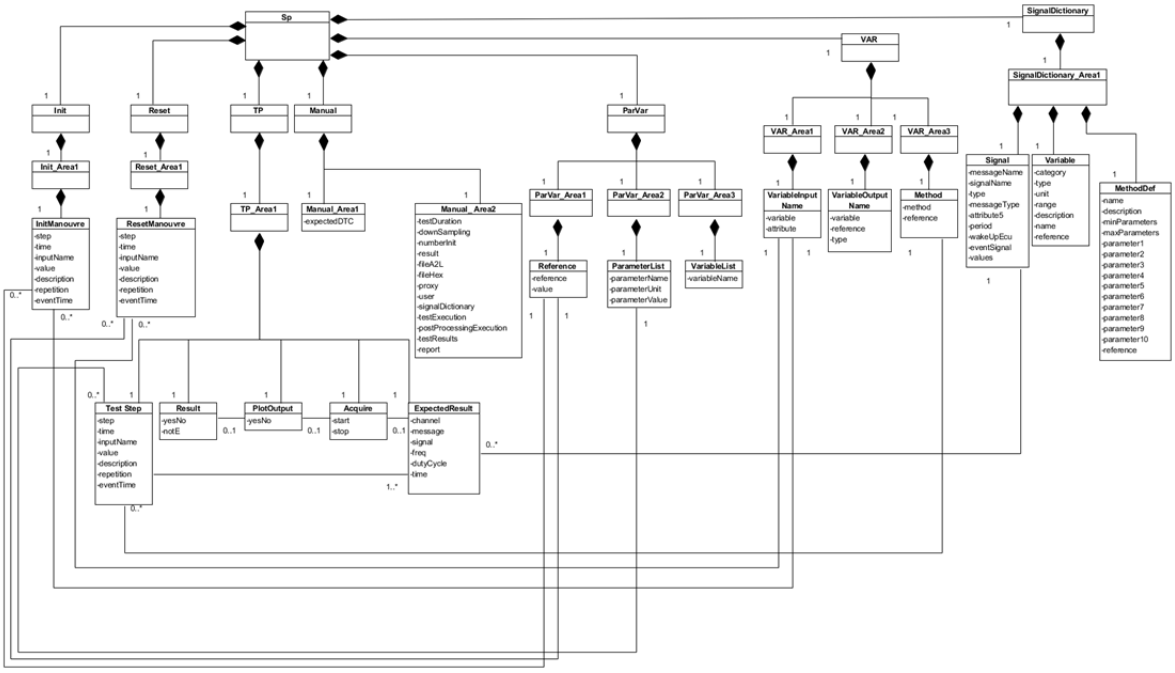

Figure 11: Inferred conceptual UML class diagram for the Test Pattern Information System.

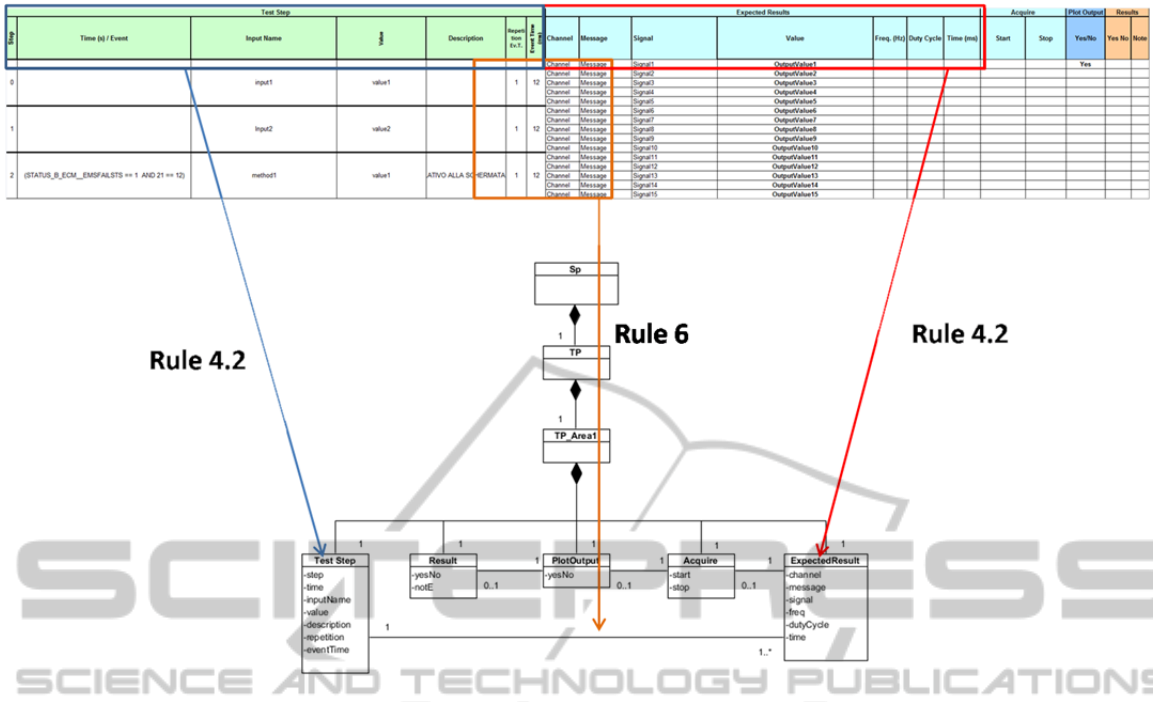

Figure 12: Example of class proposal and its mapping with the spreadsheet file.

As to the rules regarding the identification of classes, we observed that they were always successful except for three cases. In two cases, the expert decided to merge two candidate classes into a single one. These classes had been considered as different by *Rule 4.2*, since they derived from two subareas having headers with different colors but actually belonging to the same concept.

In a single case the expert decided to extract an additional class from a candidate one and assigned it a subset of the attributes of the inferred one. In this case the Rules 4.2, 5 and 6 were not able to identify this extra class since the cells associated to these two concepts had the same formatting and layout style. As a consequence, our process was not able to discriminate between them. In both cases the process failed because the spreadsheet did not comply with the formatting rules exploited by our process.

As to the rules we used to associate the candidate classes with a name, only in 21 over 110 cases they failed, since the spreadsheets did not include meaningful information to be exploited for this aim.

Furthermore we observed that in some cases the expert decided to discard some of the proposed composition relationships between the inferred classes. This fact occurred when the proposed conceptual class diagram presented a particular pattern like the one showed in Figure 13. In this case, the expert decided to move the attributes of the leaf class (*Sheet_SubArea1*) to the *Sheet1* class and to remove the remaining classes, as shown in Figure 14. This specific pattern occurred 8 times and in 6 cases the expert decided to make these changes. This result showed us the need to introduce new rules aimed at reducing the complexity of the class diagram that may be used in the occurrence of this particular pattern.

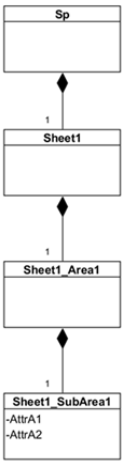

Figure 13: Conceptual class diagram inferred from a sheet having a single area containing only one subarea.

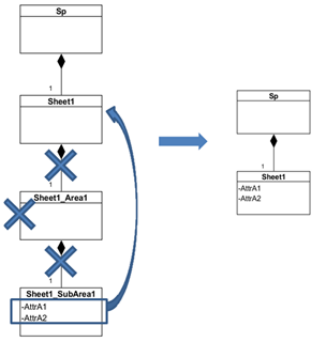

Figure 14: Class Diagram reduction proposed by the domain expert.

5 CONCLUSIONS

In this paper we presented a process for inferring a conceptual data model from a spreadsheet-based information system. The process has been defined in an industrial context and validated by an experiment involving three different spreadsheets-based information systems from the considered automobile industrial domain. The results of the experiment showed the applicability of the process and the acceptability of the inferred models, according to the judgment of experts about the application domain of the spreadsheets.

Our work differ from other ones described in the literature, since the proposed approach has been tailored to spreadsheets used to implement information systems, rather than calculation sheets.

In future work, we plan to perform further experimentations involving other spreadsheets-based systems belonging to different application domains. Moreover, we want to extend our approach further, by proposing reverse engineering techniques aimed at inferring the functional dependencies between the data embedded in the spreadsheets by analyzing the VBA functionalities they include.

ACKNOWLEDGEMENTS

This work was carried out in the context of the research project IESWECAN (Informatics for Embedded SoftWare Engineering of Construction and Agricultural machiNes - PON01-01516), partially founded by the Italian Ministry for University and Research (MIUR).

REFERENCES

Abraham R. and Erwig M., Header and unit inference for spreadsheets through spatial analyses. In Proceedings of the IEEE International Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2004, pages 165–172.

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

Abraham R., Erwig M. and Andrew S., A type system based on end-user vocabulary. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Washington, DC, USA, IEEE Computer Society, 2007, pages 215–222.

Abraham R. and Erwig M., Mutation operators for spreadsheets. IEEE Transactions on Software Engineering, 35(1):94–108, 2009.

Ahmad Y., Antoniu T., Goldwater S. and Krishnamurthi S., A type system for statically detecting spreadsheet errors. In Proceedings of the IEEE International Conference on Automated Software Engineering, 2003, pages 174–183.

Amalfitano D., Fasolino A.R., Maggio V., Tramontana P., Di Mare G., Ferrara F., Scala S., Migrating legacy spreadsheets-based systems to Web MVC architecture: An industrial case study, Proceedings of CSMR-WCRE, 2014, pages 387-390.

Amalfitano D., Fasolino A.R., Maggio V., Tramontana P., De Simone V., Reverse Engineering of Data Models from Legacy Spreadsheets-Based Systems: An Industrial Case Study, Proceedings of the 22nd Italian Symposium on Advanced Database System, 2014, pages 123-130.

Bovenzi D., Canfora G., Fasolino A.R., Enabling legacy system accessibility by Web heterogeneous clients. In proceedings of the Seventh European Conference on Software Maintenance and Reengineering, IEEE CS Press, 2003, pages 73-81.

Canfora G., Fasolino A.R., Frattolillo G., Tramontana P., A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. Elsevier, Journal of Systems and Software, 2008, vol. 81(4):463–480,

Chen Z. and Cafarella M., Automatic web spreadsheet data extraction. In Proceedings of the 3rd International Workshop on Semantic Search Over the Web (SS@ '13). ACM, New York, NY, USA, 2013, 8 pages.

Cunha, J., Erwig M., Saraiva J., Automatically Inferring ClassSheet Models from Spreadsheets. In IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE CS Press, 2010, pages 93-100.

De Lucia A., Francese R., Scanniello G., Tortora G., Developing legacy system migration methods and tools for technology transfer. In Software Practice and Experience 38(13), Wiley,2008, pages 1333-1364.

Di Lucca G.A., Fasolino A.R., De Carlini U., Recovering class diagrams from data-intensive legacy systems. In

Proceedings of International Conference on Software Maintenance, ICSM, IEEE CS Press, 2000, pages 52-62.

Fisher M. and Rothermel G., The EUSES spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In In 1st Workshop on End-User Software Engineering, 2005, pages 47–51.

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

Hermans F., Pinzger M. and van Deursen A., Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of the 33rd International Conference on Software Engineering* (ICSE '11). ACM, New York, NY, USA, 2011, pages 451-460.

Hung V., Benatallah B. and Saint-Paul R., Spreadsheet-based complex data transformation. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (CIKM '11). ACM, New York, NY, USA, 2011, pages 1749-1754.

Janvrin D. and Morrison J., Using a structured design approach to reduce risks in end user spreadsheet development. Information & Management, 37(1):1–12, 2000.

Mittermeir R. and Clermont M., Finding high-level structures in spreadsheet programs. In Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE), IEEE Computer Society,2002, pages 221-232.

Panko R.R. and Halverson R.P., Individual and group spreadsheet design: Patterns of errors. In Proceedings of the Hawaii International Conference on System Sciences (HICSS), 1994, pages 4–10.

Ronen B., Palley M.A. and Lucas H.C., Spreadsheet analysis and design. Communications of the ACM, 32:84–93, 1989.

Scaffidi C., Shaw M., Myers B., Estimating the Numbers of End Users and End User Programmers. In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, 2005, pages 207-214.

Shokry H., Hinchey M., Model-Based Verification of Embedded Software. In IEEE Computer, 42(4), 2009, pages 53-59.