Systematic Spreadsheet Construction Processes

Jorge Mendes*, Jácome Cunha†, Francisco Duarte‡, Gregor Engels§, João Saraiva* and Stefan Sauer§
*HASLab, INESC TEC & Universidade do Minho, Portugal, email: {jorgemendes,saraiva}@di.uminho.pt
†NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa, Portugal, email: jacome@fct.unl.pt
‡Bosch Car Multimedia, Portugal, email: Francisco.Duarte@pt.bosch.com
§Universität Paderborn, Germany, email: {engels,sauer}@upb.de

***Abstract*—Spreadsheets are used in professional business contexts to make decisions based on collected data. Usually, these spreadsheets are developed by end users in an ad-hoc way. Thus, the business logic of a concrete spreadsheet is not explicit to end users, making its correctness hard to assess and users have to trust.**

We present an approach where structure and computational behavior of a spreadsheet are specified by a model with a process-like notation based on pre-defined functional spreadsheet services with typed interfaces. This enables a consistent construction process of a spreadsheet that comprises defining its structure and computational behavior as well as filling it with data and executing the defined computational behavior. Thus, concrete spreadsheets are equipped with a specification of their construction process. This supports their understanding and correct usage, even in case of legacy spreadsheets.

The approach has been developed in cooperation with an industrial partner.

***Keywords*-model-driven engineering; situational method engineering; construction process; spreadsheet**

I. Introduction

Spreadsheets are used in professional business contexts to make calculations and decisions based on collected data: it is estimated that 95% of all U.S. organizations use spreadsheets for financial reporting [16], 90% of all analysts in industry perform calculations in spreadsheets [16], and 50% of all spreadsheets are the basis for decisions [12]. Spreadsheets are not only used to define sheets containing data and formulas, but also to collect information from different systems, to adapt data produced by one system to the format required by another, to perform operations to enrich/simplify data, to present data in graphic (visual) representation, etc.

It has been observed that despite admitting serious risks, many organizations manipulate large spreadsheets in a frightening ad-hoc way: they are adapted/enriched/evolved by using an unspecified/undocumented process, usually performed by users directly updating the computational structure or data of a spreadsheet. This is even more frightening as the business logic of a concrete spreadsheet is hidden, baffling and difficult to understand by end users. Thus, the correctness of a spreadsheet is hard to assess and users have to trust.

A careful analysis of this situation reveals a few shortcomings in the ad-hoc construction of spreadsheet applications:

- Spreadsheets do not have the notion of a type level. Thus, spreadsheets are developed on the instance level where no type checking is possible. This leads quite often to structural changes, like insertion of rows and columns, that are not complete and consistent as certain dependencies are only implicit and cannot be detected by the spreadsheet system. This situation occurred in the spreadsheet used for an economical study on the level of austerity that countries should comply with [2].
- The missing concept of types can lead to inconsistent computation sequences where the data flow between two computation steps is not appropriate [11].
- The internal data and control flow of computation steps within a spreadsheet is only implicitly defined by a spreadsheet developer [11]. The missing explicitness and documentation hinders any kind of traceability and understanding of a given, often legacy spreadsheet as well as any kind of maintenance of a spreadsheet [11].

There exist several approaches in the literature to overcome these issues in other contexts. We combine them in a new way proposing a novel solution which enables precise, understandable, and repeatable construction of spreadsheets.

First, the concept of ClassSheets has been previously introduced, which allows developers to define the logic and structure of a spreadsheet on the type level [7]. Although several extensions have been proposed such as types for cell values [3] or embedding ClassSheet models in a spreadsheet hosting system [4]–[6], so far only the specific part of the spreadsheet cells and formulas has been targeted, leaving out several features commonly used in industrial situations.

Secondly, Model-Driven Engineering (MDE) has been introduced as a methodology to specify structural and behavioral aspects of a system on an abstract model level, before it is implemented [13]. This is also applied for process-modeling tasks, e.g. in business or production process modeling. Having a fully-fledged process also allows us to enact it and to automate the execution of individual steps of the process.

Thirdly, Situational Method Engineering (SME) is an approach to define processes for a certain development task by composing predefined parameterized building blocks with typed interfaces to a consistent process [10]. In this context, the meta-model based approach MetaMe [8] was developed to yield sound and typed methods. Based on this, the framework MESP (Method Engineering with Method Services and Method Patterns) [9] was built. The basic idea is that methods should be composed from method services.

The combination and adaptation of these three approaches

978-1-5386-0443-4/17/$31.00 ©2017 IEEE

yields our novel and integrated approach to define the *construction process of spreadsheets* for business applications. This approach consists of a process (on the meta-level) for creating and executing the spreadsheet construction process based on elicited requirements (see Fig. 1). This process comprises (a) defining the structure and computational behavior of a spreadsheet (construction process design) as well as (b) filling it with data and executing the defined computational behavior (construction process enactment). Thus, the basic idea of our approach is to equip a spreadsheet with an operational specification based on *functional spreadsheet services*. These functional services are inspired on the method services introduced before. This yields consistent spreadsheets with a documented internal computation structure.

The (meta-)process of creating and executing a construction process for spreadsheets is organized in three distinct phases as illustrated in Fig. 1.

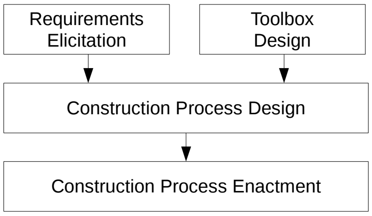

Fig. 1: A three layers approach for the construction process of spreadsheets.

In order to validate the proposed spreadsheet construction process, we consider a large and complex industrial case study: the Bosch Car Multimedia quality report spreadsheet-based system. In this system large amounts of data are collected from two SAP enterprise resource planning (ERP) systems. The data is then manually transformed by Bosch engineers so that a monthly report is delivered to the administration. In this paper, we show how such an ad-hoc, error-prone and time-consuming task is documented/specified and automated using our approach. Although our approach has been developed and evaluated in close collaboration with Bosch engineers to fulfill their requirements, we plan to further demonstrate its validity with other industrial partners.

II. Spreadsheet Construction Framework

The framework we propose aims at providing a safe way for designing and constructing spreadsheets. The key concepts behind it are the specification of the actions that are to be undertaken to create and use a spreadsheet, which we call *functional services*, and the artifacts and their *types* that are required by or originate from these actions.

The functional services that are referred in this work are the ones that users have available in common spreadsheet systems (e.g. insert a pivot table). This work provides a way to specify actions for existing features, but also for new features that may be introduced in the future in spreadsheet systems by proving generic ways of defining them. The functional services are specified as actions (in UML activity diagrams [1], [15]) that can receive artifacts as input and can produce artifacts as output.

The artifacts are typed, that is, we make a distinction between any two artifacts that are not compatible with each other or that do not serve the same purpose. For instance, a *chart* is a different type than a *pivot table*. This allows us to restrict the inputs to the actions that are to be performed, since they usually only work on specific artifacts.

The actions and artifacts are combined in a UML activity diagram, indicating the control and data flows of the spreadsheet construction process.

The three phases of the (meta-)process for defining and executing the spreadsheet construction process are supported respectively by a three layers architecture (Fig. 1):

- requirements elicitation and toolbox design,
- construction process design, and
- construction process enactment.

The first phase has two different tasks. First, there is the common requirements elicitation for the spreadsheet to be produced. Since this is a process well studied, we will not discuss it further. Second, the toolbox design should be performed by experts in information systems. In this step the functional services are identified as well as their input and output (types). We provide a first version of the toolbox which can later be extended. In the next phase – construction process design – functional services are instantiated for the spreadsheet under consideration. This step should not be done by spreadsheet end-users, but by domain experts trained to use the toolbox content. Finally, the construction process enactment can be executed whenever necessary either automatically by the spreadsheet system, if possible, or assisting a spreadsheet end-user by providing the required parameters.

In the following sections we detail each of these three steps.

A. Toolbox Design

The toolbox provides the process designer (the one responsible for creating a particular spreadsheet construction process) with a set of tools to specify a concrete construction process. It is composed of a set of functional services and a set of types of the inputs and outputs of the functional services.

1) Functional Services: Functional services are defined based on the features that spreadsheet systems provide. The types and parameters are gathered from the dialogues presented to the users or from the APIs where they are detailed. In this case, we refer to the most common spreadsheet features available in most spreadsheet systems such as: add a new column or row, delete a value from a cell, or modify a formula. Some of these features have interactions with the outside world, e.g., input by the user, or read the content of a file.

A functional service defines a step or composition of steps to be performed within a spreadsheet. It performs an action with given input and output types, but also with other parameters required for such an action. For instance, importing a CSV file into a ClassSheet table is defined by the functional service defined in Fig. 2. In this case the functional service requires two inputs: one is the table (*import_to*) to import the data to,

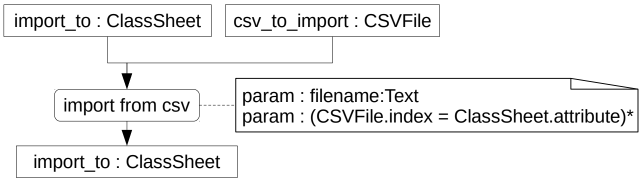

Fig. 2: Activity diagram for the *import from csv* functional service.

defined by a ClassSheet, and the other is the format of the CSV file (*csv_to_import*) specifying the columns in the file. This functional service requires two extra parameters: one being the CSV file itself so the spreadsheet system can import it, and the other being a mapping between each column of the CSV file and each attribute of the table given by the ClassSheet. In this case the output produced is again the table but filled with the data imported from the CSV file.

Note the difference between inputs and parameters. The inputs are used to guarantee that the functional service is only coupled with other functional services that produce those artifacts as output. The parameters are extra information necessary to configure the service, as requested by the dialogues of the spreadsheet system.

2) Types: In our framework we provide base types available in spreadsheet systems as numbers; text; lists of elements of some type (e.g. a list of labels); references; (data) tables, specified using ClassSheets; and generic types that can be further defined during the construction process design. ClassSheets are only able to specify the contents of the cells of a worksheet, including their position. Thus, other elements such as charts and images are in our case defined using a generic type that can be further instantiated. Generic values have a name (e.g., "chart"), an identification (*id*) that can be referenced, and a possibly empty set of attributes and corresponding values of any available type. Generic types represent the several artifacts that spreadsheets support, but they can also be used to represent input and output types of the functional services. This allows us to have a uniform representation of many of the types that are used as input and output of actions.

B. Construction Process Design

The construction and usage of a spreadsheet can be defined as a workflow using the previously described functional services. All actions to be performed are specified using functional services, which define the kind of interaction the end user should perform with the spreadsheet and thus how the spreadsheet should be used.

By defining a workflow we want to give some correctness guarantees, namely that the sequencing of functional services is type safe, i.e., the output type of a functional service matches the input type of the following one. This deviates from the common usage of spreadsheets which allows for a very flexible usage of many of its components, but can lead to errors in intermediate steps.

A spreadsheet construction process model (being the outcome of this construction process design phase) is used to specify the steps to obtain a spreadsheet from the provided requirements. The designer has to insert into the process the available inputs and define the expected outputs. Then, the intermediate actions to obtain the required outputs from the available inputs have to be defined.

The design process is restricted by the types of the intermediate actions, which prevents a class of errors and ensures a safe flow of data. Moreover, the use of types allows us to specify exactly which kind of data or artifact is provided by some action or is given to another. In some cases, explicit typing may not be required and thus a functional service is connected directly to another.

C. Construction Process Enactment

The enactment of a construction process consists of performing the actions defined by the functional services in that process and in the order they are specified. Whenever some required parameter of a functional service is not defined, the end user enacting the process is prompted to provide the value. User interaction may also happen when an action explicitly requires input from the user (e.g., filling a table). For instance, consider the *import from csv* functional service. To enact this service the spreadsheet user needs to actually select a CSV file so the process can continue.

Nevertheless, most of the process should be automated to prevent human errors. We plan to implement a tool that can automate the enactment of a construction process as much as possible without human intervention. The use of a standard UML language to represent the process itself will be useful as this allows us to use available tools to manipulate this kind of representations, such as the ones in the Epsilon family [14].

III. Industrial Case Study

A quality department at Bosch Car Multimedia Portugal uses a spreadsheet to analyze the data exported from a SAP ERP system. This data contains the reported defect claims of the parts they produce. In addition to the defect claims, they require information about the parts involved which is obtained from a different SAP ERP system. The data is analyzed and summarized to create a report for the stakeholders.

The analysis process is performed once a week, and the report is sent to Bosch's administration once a month. A similar report in the same format must be sent to the administration by two other Bosch Car Multimedia factories in other countries (China and Malaysia). However, the process for obtaining the report is defined by each factory individually provided that the result fits the administration's requirements.

A. Description of the Legacy Process

The analysis is spread across eight worksheets to gather the input, perform the analysis, and to report the results (Fig. 3).

Worksheets *parts* and *0km* contain the input data that is extracted from two different systems. Both of these worksheets contain database-like tables: the first row contains the descriptions of the columns while the remaining rows contain the data where each row is a record. The worksheet *parts* has 4 columns and about 1400 entries, while *0km* has 72 columns and about

2300 entries. The data in *0km* is filtered to obtain only the records for the year being analyzed (about 500 entries) and those records are copied to *format*.

The worksheets *claimed*, *customer*, and *production* are used to analyze the data from the *format* worksheet using the same process but with distinct parameters. First, a pivot table is created, then the data is copied next to the pivot table, and finally an advanced sort is applied to the data. This sort is not directly appliable to the pivot table.

A selection of the results from the three analysis worksheets is grouped in the worksheet *summary*, with a better visualization of that worksheet provided in the *result*.

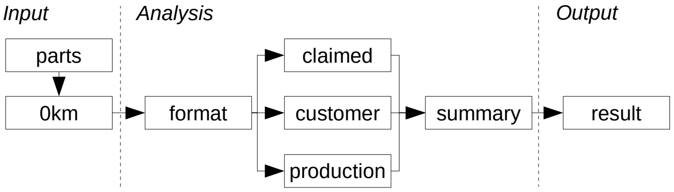

Fig. 3: Worksheets data dependency.

B. Specification of the Construction Process

The legacy process described in the previous section can be implemented in our framework using the functional services that we provide. Due to length constraints, the full process is not detailed in this paper. However, it fully describes the legacy process, as depicted in Fig. 4, and provides a completely automatic approach to create this weekly spreadsheet.

C. Discussion

The process described in this industrial case is mechanical, tedious, and error prone. Bosch employees spend a considerable amount of time in this process to create a report every week, which could be better spent doing in-depth analyses.

Our framework lifts the burden to manually create the spreadsheet. At this stage, it provides an explicit sequence of steps to recreate the spreadsheet whenever it is necessary (e.g. when initial data changes). In the future, all this process can be automated and thus freeing employees from this task.

IV. Conclusion

In the past, it has been often reported that the use of spreadsheets in any kind of business application is of high risk. This is (also) due to missing structured development processes of spreadsheets as well as due to missing quality assurance activities in ensuring the quality of spreadsheets. Quite often, spreadsheet end-users do not question the correctness of spreadsheets and come to decisions with high impact based on spreadsheet analysis even if they do not fully understand the underlying functionality of the spreadsheet.

In this paper, we present a novel approach to make the structure and computation flow within a spreadsheet explicit by a construction process. These construction processes are composed of predefined parameterized and typed functional services, which allows us to guarantee their correct composability. The construction process also allows us to automate the creation and usage of the spreadsheet.

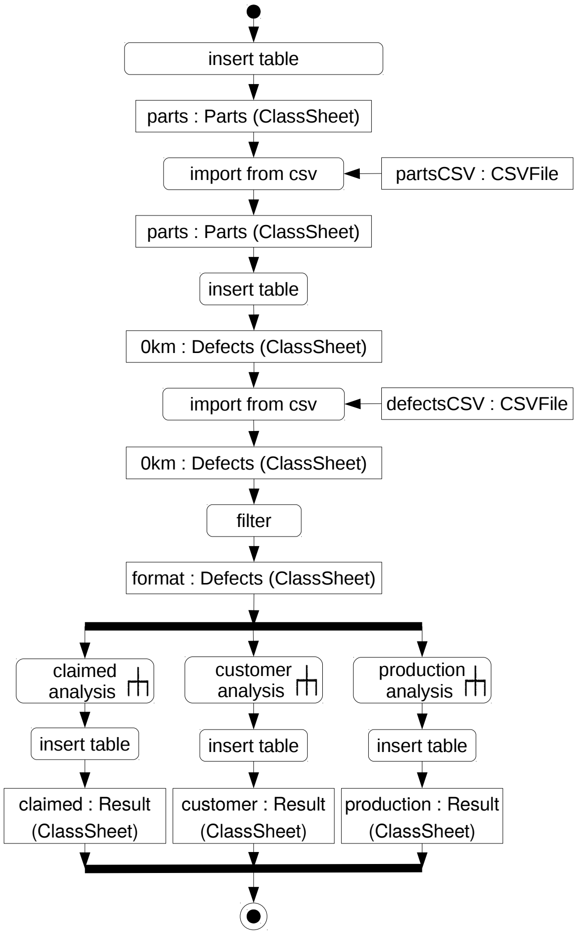
insert table
parts : Parts (ClassSheet)
import from csv
partsCSV : CSVFile
parts : Parts (ClassSheet)
insert table
0km : Defects (ClassSheet)
import from csv
defectsCSV : CSVFile
0km : Defects (ClassSheet)
filter
format : Defects (ClassSheet)
claimed analysis
customer analysis
production analysis
insert table
insert table
insert table
claimed : Result (ClassSheet)
customer : Result (ClassSheet)
production : Result (ClassSheet)

Fig. 4: Bosch's spreadsheet construction process model.

The presented approach has been developed in collaboration with an industrial partner, whose spreadsheet construction process and dependency structures have been made explicit with our construction process model, as discussed before.

The next steps are to enlarge the case studies and to reengineer existing spreadsheets used in industrial applications. This will be done in close cooperation with industrial partners from the Software Innovation Campus Paderborn (SICP). It is expected that this might lead to additional functional services in the provided toolbox. This research will be accompanied by the development of a tool for further support.

Acknowledgment

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project POCI-01-0145-FEDER-016718, and by FCT as part of project UID/CEC/04516/2013. This work is also financed by the bilateral project FCT/DAAD with ref. 441.00. The first author is funded by FCT grant SFRH/BD/112651/2015.

References

[1] J. Arlow and I. Neustadt. *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2Nd Edition)*. Addison-Wesley Professional, 2005.

[2] P. Coy. Faq: Reinhart, rogoff, and the excel error that changed history. Bloomberg: http://www.bloomberg.com/news/articles/2013-04-18/faq-reinhart-rogoff-and-the-excel-error-that-changed-history, 2013 April.

[3] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Extension and implementation of classsheet models. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 19–22, Sept 2012.

[4] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. MDSheet: A Framework for Model-driven Spreadsheet Engineering. In *Proceedings of the 34rd International Conference on Software Engineering*, ICSE'12, pages 1395–1398. ACM, 2012.

[5] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Embedding, evolution, and validation of model-driven spreadsheets. *IEEE Transactions on Software Engineering*, 41(3):241–263, March 2015.

[6] J. Cunha, J. Mendes, J. a. P. Fernandes, and J. a. Saraiva. Embedding and evolution of spreadsheet models in spreadsheet systems. In *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '11, pages 186–201, Washington, DC, USA, 2011. IEEE Computer Society.

[7] G. Engels and M. Erwig. Classsheets: Automatic generation of spreadsheet applications from object-oriented specifications. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 124–133. ACM, 2005.

[8] G. Engels and S. Sauer. *A Meta-Method for Defining Software Engineering Methods*, pages 411–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] M. Fazal-Baqaie and G. Engels. *Software Processes Management by Method Engineering with MESP*, pages 185–209. Springer International Publishing, Cham, 2016.

[10] B. Henderson-Sellers, J. Ralyté, P. J. Ågerfalk, and M. Rossi. *Situational Method Engineering*. Springer, 2014.

[11] F. Hermans. *Analyzing and visualizing spreadsheets*. PhD thesis, Delft University of Technology, 2012.

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

[13] S. Kent. *Model Driven Engineering*, pages 286–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[14] D. Kolovos, L. Rose, R. Paige, and A. Garcia-Dominguez. *The Epsilon Book*. Eclipse, 2010.

[15] Object Management Group. UML specification 2.0, 2005. available at http://www.omg.org/spec/UML/.

[16] R. R. Panko and N. Ordway. Sarbanes-oxley: What about all the spreadsheets? *CoRR*, abs/0804.0797, 2008.