End-User Development Framework with DSL for Spreadsheets

Vineta Arnicane

University Of Latvia, Faculty of Computing, Raina blvd. 19, Riga, Latvia
Vineta.Arnicane@lu.lv

Abstract. We propose a framework that encourages involving of end-users in software system development as designers of executable interface specifications. The framework is intended for development of information systems with spreadsheets in role of user interface. A domain specific language allows describing the necessary actions in spreadsheets and ties the spreadsheets to system data using data ontology. We have developed a program that translates a template into a spreadsheet together with customized operations for data saving, updating or retrieving. Our experience gained during the development of two software systems based on the proposed framework is described in this paper.

Keywords: End-user software engineering, end-user programming, spreadsheet, data ontology.

1 Introduction

There is a group of institutions that need to store and analyze data submitted as documents consisting primarily of tables; the analysis reports sought in most cases are also tabular data. Due to changes in business environment and user demands, the requirements for information systems change very often. New documents have to be accepted by a system as input data and new types of reports are necessary to support the users' business decisions. It is an expensive process to maintain a continuously and rapidly evolving software system. We propose a framework that is intended to involve the end-users in the process of software development as much as possible.

Users like their habitual software environment, they like it when a new system's interfaces are easy to understand, comfortable and learnable very. On the other hand, the users demand that developers implement new requirements quickly, and that systems are reliable. The users want to be able to obtain data from systems quickly and with little effort in order to make business decisions.

The users are better experts on user interfaces – they more or less exactly know what the reports have to look like and which kind of input forms they prefer [1]. The users often use spreadsheets to create specifications for the tabular-data containing documents they need in the system.

As a consequence, our framework offers to use spreadsheets as the basis for the user interface for a couple of reasons. Firstly, the users are familiar with spreadsheets due to their widespread use in office software. Secondly, the data in report

spreadsheets are well suited for further processing – for example, in pivot tables, charts, word processors. Thirdly, professional software developers are exempted from user interface development. If the end-users work is of a good quality level then this approach saves a lot of time for professionals.

End-user programming is practiced on a large scale with the aim to write programs to support goals in their own domains of expertise [2]. End-user development has been defined as "a set of methods, techniques, and tools that allow the users of software systems, who are acting as nonprofessional software developers, at some point to create, modify, or extend a software artifact" [3]. In our case the users create spreadsheet templates using a very simple visual domain specific language [4]. Templates can be considered programs, if programming is defined as in modern English dictionaries – the process of planning or writing a program, where program is defined as a collection of specifications that may take variable inputs, and that can be executed (or interpreted) by a device with computational capabilities [2].

Research shows that spreadsheets are widely used in end-user programming [5-9]. Erwig et.al. [10] present a program generator that translates a template into a spreadsheet ready for use together with customized update operations for changing cells and inserting/deleting rows and columns for this particular template. Their program isolates the user from mistakes made while filling the spreadsheet with data by assisting the user in inserting or deleting rows or cells in the spreadsheet.

Our framework also supports the saving of data in a database and retrieving data from it. All data descriptions are registered in the system's ontology. The ontology defines a set of representational primitives usable in modeling a domain of knowledge, where the representational primitives are typically classes, attributes, and relationships [11].

The ontology supported by our framework contains information about the kinds of documents acceptable to the system via its interfaces as well as descriptions for each data item in each kind of input documents that are stored – numbers, strings. There are also data quality rules and integrity controls stored in the ontology. These rules represent the relationships between data items stored in the system.

The data descriptions stored in the ontology are used to refer to the data in spreadsheet templates using our domain specific language and they are used in database stored procedures when retrieving data for complicated reports.

The framework's software supports data input via spreadsheets generated from templates, data integrity control according to the rules stored in the ontology, and retrieving the data for simple reports, generated as spreadsheets according to the given templates. For the more complicated reports, the data retrieving procedures have to written by professional programmers.

With this approach, professional programmers and system analysts perform only the more complicated tasks; most of the routine tasks can be done by end-users.

Our approach has a few advantages. Firstly, it reduces the amount of work done by professional developers; secondly, it achieves higher user satisfaction with interfaces because the spreadsheets look precisely as they were designed; and, thirdly, it reduces software testing complexity because template testing is usually simpler than testing newly programmed interfaces.

2 Information systems considered

Proposed framework is intended for supporting the information systems that accumulate information submitted from various sources as table shape documents. The reports of these systems mostly are also table shape documents. These systems have to provide checking of data mutual integrity and data quality, too.

Input documents and reports can be grouped in four groups. The first group contains documents with *fixed shape* (Fig.1a). Often these documents are tables with predefined fixed count of rows and columns with data. Next three types of documents are with *flowing shape*. Documents of the second group contains table with fixed header and with unknown count of rows (Fig.1b). Each row contains the information of the same type. The third group is similar to second with rows and columns in mutually interchanged roles – the count of rows is fixed but the count of columns is unknown during specification of report (Fig.1c). Finally, the fourth group consists of documents with unknown at the time of specification count of rows and columns (Fig.1d).The real count of rows and/or columns in reports is determined either by data, for instance, as clients in reports shown in Fig.1b and Fig.1d or values of report controls chosen by user as shown in cases if Fig.1c and Fig.1d.

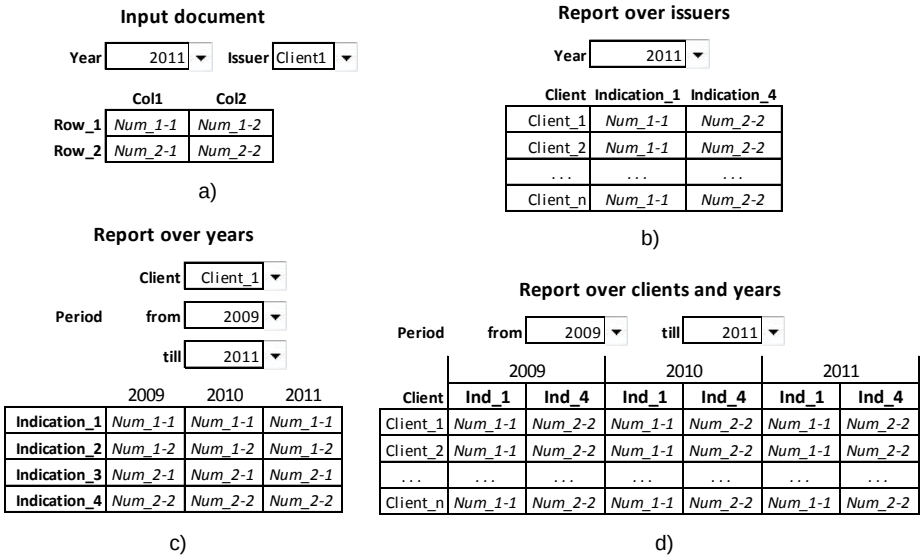

Fig. 1. Schematic sample of input document (a), different kinds of reports – vertically flowing report (b), horizontally flowing report (c), horizontally and vertically flowing report (d).

Input documents usually fall into first and second group. The documents of third and fourth groups typically are reports.

Each document has two main parts. The first part contains information about the whole document, for instance, – issuer, the period about it contains the information. The second part contains data values, for instance, different figures or indications and descriptions of these figures, for instance, the headers of columns or titles of rows.

As schematic sample let's consider an input document with fixed shape as shown in Fig.1a. It is submitted by each issuer yearly. It contains four indications Indication1, Indication2, Indication3 and Indication 4 accordingly coded as Num_1_1, Num_1_2, Num_2_1, Num_2_2. Indications can be, for instance, profit earned during corresponding year or taxes paid. There are actual figures replaced with their data idem codes in Fig.1a. There can be made three kinds of reports based on these indications – (1) over issuers for specific year like one shown in Fig.1b, (2) over years (Fig.1c) or over issuers and years (Fig.1d).

For instance, if look closer to report in Fig.1b we can see that there are the same data item codes for every client in each column. There is not a mistake. These are codes of data items. If it would be a real report in place of each code would be the appropriate number each client submitted for year 2011.

3 System architecture

Proposed system architecture consists of four main parts - data ontology, data storage, front-end and back-end software (see Fig. 2).

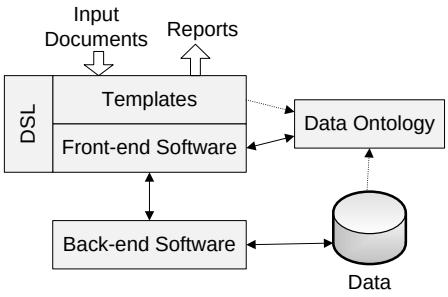

Fig. 2. Architecture of system developed on basis of framework.

Ontology describes system's business data. All input data in systems are coming in chunks called as documents. Each document has a set of data items. Ontology contains information about document types as well as types of data items and rules of data integrity and consistency.

Data storage holds system's business data – entered documents and values of their data items.

Front-end software is composition of standard spreadsheet software, system's templates, and framework's software that realizes the interpretation of templates specified using framework's custom DSL.

Back-end software ensure security issues during work with data storage, various data wrappers for access data in data storage, stored procedures of data base written with aim to retrieve data for the most complicated reports.

4 System data ontology

Business data ontology aggregates the information about types of input documents that are stored in system in common with types of business data. It contains also information about types of business data reports that can be generated from system. Each type of document has a set of attributes, for instance, what is document's type periodicity – year, quarter or moth, which institution or group of institutions has to submit documents of this type, dates from when and till when document type is valid for submission and also the name of spreadsheet template that supports the entering and modifying of business data.

The more interesting information in ontology is about data item types. Each document type contains one or more data item type. There is a set of attributes stored in ontology for each data item type, for instance, reference to document type to which item belongs, item code, item description and also the date from when and till when item type is valid. Fig.3 shows the conceptual schema of data ontology.

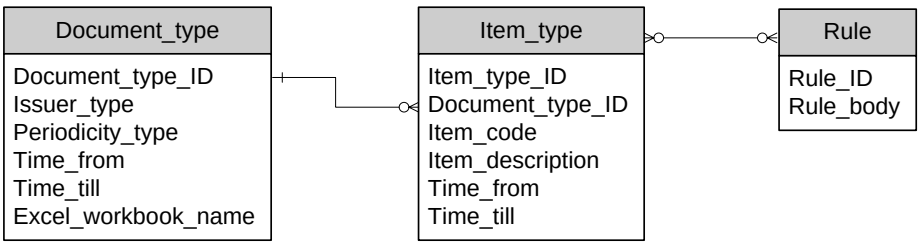

Fig. 3. Conceptual schema of data ontology.

The code of each data item type is unique within the scope the information system. This feature allows easy and precisely refer to data items in system's user interface templates as well as in rules for checking the data quality and consistency.

5 DSL for spreadsheet templates of user interface

Our purpose was to give to end users a tool that allows them to make templates for their business systems by own as much as it is possible. Let's look on main features of spreadsheets that framework's software put to use when interprets documents template.

5.1 Main spreadsheet features used in DSL

Range is cell of spreadsheet (Fig.4a) or contiguous group of cells (Fig.4b). Ranges can be or labeled, for instance, Cost as shown in Fig.4a. Range can be referenced as addresses of cells, for instance, as in formula shown in Fig.4b.

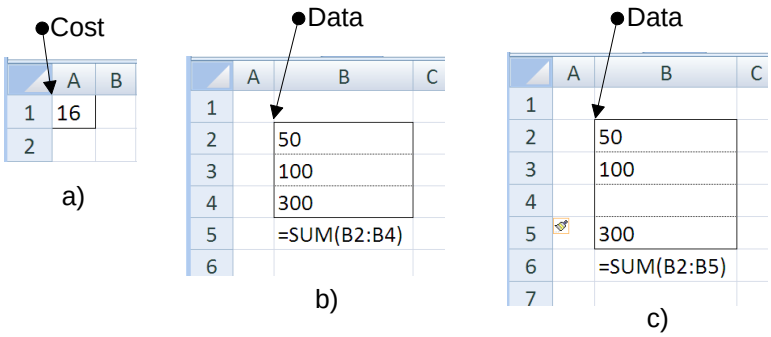

Fig. 4. Samples of ranges – (a) range with single cell, (b) range as contiguous group of cells, (c) range as contiguous group of cells after insert of row before last cell

If range is contiguous group of cells it stretches when new cells are inserted between every two cells belonging to range as it is shown in Fig.4b and Fig.4c. There are range Data and formula referring to the range shown in Fig.4a. When new cell is inserted after cell B4, spreadsheet's functionality automatically ensure that label Data now refers to its own cells including new cell as shown in Fig.4b. What is also very important – formulas referring to ranges by relative addresses also automatically are updated according to new situation, for instance, as shown in figures Fig.4b and Fig.4c. Absolute addresses used in formulas remain unchanged.

5.2 DSL elements

The elements of our DSL are interpreted by spreadsheet macros or programs created in another environment that supports manipulation of spreadsheet using API. All DSL elements are labelled ranges. The advantage is that end users can make the spreadsheet by principle What You See Is What You Get. Our DSL is very small. It has only few language elements that can be grouped as:

- Static elements, for instance, – the title of document, header of table that contains data items.
- Control elements – to specify the values of document attributes according to its type, for instance, to dropdowns for year, year and quarter or year and moth according to periodicity of the type of document specified in template, dropdown to choose the name of issuer.
- Data elements – spreadsheet's cells where the data will be entered or modified.
- Other DSL specific elements – labelled cells and ranges, hidden sheets with technical data – months, quarters, years, elements of SQL query, data integrity checking rules.

Usage of these elements is so simple that it can be done by end-user. For instance, let's look on some of them in template of fixed shape input document shown in Fig.5a.

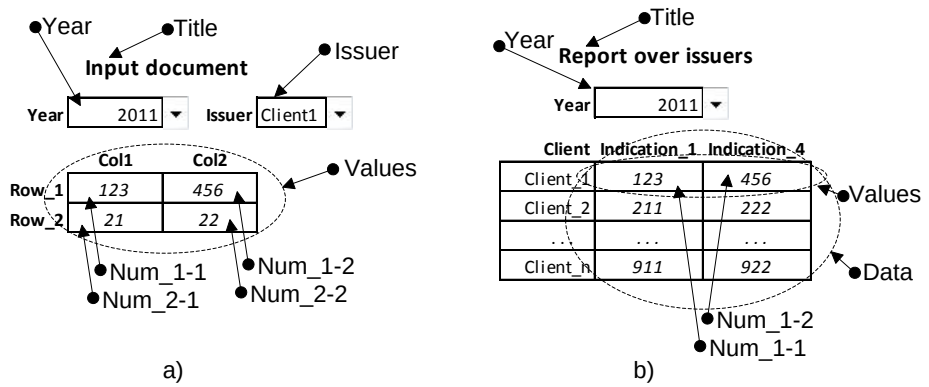

Fig. 5. Samples of document templates with indicated ranges – (a) template of fixed shape document, (b) template of vertically flowing shape document.

Static elements are ordinary strings, numbers or other values in labelled spreadsheet cells that are only read by interpretator of our DSL. They never are changed. For instance, range Title in Fig.5a is static element.

Control elements are ranges Month and Issuer. These elements look as ordinary cells but when user tries to change their values he or she can only choose one element from the predefined list. Creator of template copies the values for lists of control elements in hidden sheet in template's workbook.

Data items are ranges consisting of single cell, but whole data items in template are surrounded by range Values. Software knows that al data are in range Values but label of each individual cell indicates data item that should be showed in it or read from it.

Flowing shape templates are more complicated. Let's look on vertically flowing report as shown in Fig.5b. They have range Values included in range Data. Range Values forms the second row of range Data. Range Data consists of range Values and one empty row before it and another empty row after it. Each time when it is necessary new line is added in range Data before the last row and it is filled accordingly data item places in range Values.

The templates for horizontally flowing shape documents are analogue to templates of vertically shape documents only there work is done with columns instead of rows.

Using the same principle the templates for horizontally and vertically flowing shape documents can be specified and interpreted.

Framework supports also reports with multiple groups of data items with subcalculations, for instance, subsums.

There are also some another template's ranges used by framework's software, for instance, with the aim to perform the data consistency checking, but for the sake of shortage of place we will not discuss them.

6 Case studies

Our framework has been used as basis of development of two information systems. Users of these systems collect data and analyse them for statistical needs. Users of both systems are from a governmental financial institution with some 500 employees. The organisation has a central office and few branches in various parts of Latvia.

The first of systems at this moment has 205 input document templates and 1397 report templates; the second one has 132 input document templates and 185 report templates. Both systems persistently are supplemented with new templates.

The framework is realized on basis of MS Excel for user interfaces and front-end software. As storage for data is used MS SQL data base.

All templates have been designed as spreadsheets by end-users. End-users considered that development of spreadsheet templates is not hard for them in most cases.

The easiest ones were fixed shape input document templates. In order to assign unique codes for each data element they were form according to pattern *DocumentTypePrefix_Row_Column*. Each document type has its own unique *DocumentTypePrefix*. When each template was designed front-end software registered it into database. Registration meant to make a record into appropriate table for document type and for each of data elements. All data of fixed shape input documents is stored in the same tables.

For each type of flowing shape input documents there were created its own set of tables for data storage. Tables were two – one for latest values of data elements and one for data change history. The count of flowing shape document types is small – under 20.

Simple reports were completely designed by end-users. In these cases when report runs MS Excel macros generates select query using information from ranges in template.

There were very complicated flowing shape reports that were challenge for professional programmers too. They required development of storage procedures in data base in order to select and summarize information according to different aspects.

Users are very satisfied that they receive reports and input forms exactly the same as they were specified. Developers are happy that small changes and improvements in templates always can be done by users themselves. Users were forced to achieve agreement between themselves as concerned design of user interfaces.

Software testers were satisfied that testing of interfaces become easier and faster because there are not any program code in templates.

Framework's front-end software is very complicated but problems in it come to light quickly because all interfaces are operated by the same code.

The main achievement of used framework is economy of time and efforts. Users design interfaces and get exactly what they were designed. IT people – system analysts, programmers, testers care about software and data quality as regards stored procedures, MS Excel macroses and assistance in management of data elements' codes.

First prototype of front-end software was developed by one programmer/system analyst. Afterwards was created a development team consisting of two persons. One of them improved and maintained front-end software while another trained users how

to design templates, tested system, and enhanced its requirements. The first system mostly has fixed shape input documents and reports, and simple horizontally or vertically flowing type reports. At very beginning only two of input documents were of flowing type and only 5 reports were complicated-with subsums or of flowing type both horizontally and vertically. Later users developed only few new flowing type input forms and complicated reports.

When in parallel of maintenance of the first system the development of the second system was started the team was supplemented by 3 persons. Due to complexity of reports of second system one of them specialized on development of stored procedures or SQL queries for reports. Another two programmed if necessary, assisted and trained users, managed code system of data elements.

7 Conclusions and future work

We have presented a framework for development of system with spreadsheets as user interfaces. The main advantage of the framework is essential involvement of end-users in process of user interface specifications.

By introducing of end-users into programming activities we have changed the development complexity in some aspects:

- The complexity of front-end software development becomes higher. Programming of back-end kernel functionality is difficult and the software is not easy maintainable.
- New interfaces can be developed easy and quickly, most of work can be done by end- users.
- Complexity of interface design and usability testing transits from IT department to the end-users.

The implementation of ontology is realized in many parts of front end and back-end software and templates: in the rules in templates, in the database, and in the stored procedures. The methodology to write and manage rules needs to be improved.

The positive aspects of proposed method and framework are following.

- The design and implementation of input forms and reports are faster:
  - Users themselves design a spreadsheet template on order to display the necessary information, to work efficiently, to print with the needed design.
  - Users and developers do not do double work to draw or design interface input forms or reports.
  - Developers make only the technical tying of templates to the back-end software. They do not program visual components that usually consume many resources.
  - Developers do not write new code for each form, but they reuse the ready ones and save time to programming and testing.
  - If the user wants to change a visual representation of any document template, then it can be done without any support of IT specialists.
- The whole system is more stable:
  - All forms exploit one and the same code, the bugs are discovered faster and corrections run for all templates.

- It is easier to maintain code by other programmers due to small amount of code and it is not too dependent on particular input forms or reports.
- The software is more scalable and transferable to different development and runtime environment.

- Users get the system that more corresponds to their needs:
- Users use the working tool that they know very well (e.g. MS Excel).
- Users discuss and design desirable interface with minimal consultations from IT specialists.
- Users get the interface that is exactly the same as it was designed by them.
- New input forms and reports can be developed quickly and usually without traditional programming;
- Testing of new interface can be performed immediately by users.

There are negative aspects too. Some front-end and back-end software parts are complex (programmed in the higher abstraction level) and are not easy understandable. Some reports might be too complex to be expressed by given DSL.

The framework and methodology is very useful for some domains of applications. The results by applying described approach were more positive then it was expected. The research is needed to extend domain of applications and improve DSL to deal with more complex requirements for user interface.

Acknowledgments. This work has been supported by the European Social Fund Project No. 2009/0216/1DP/1.1.1.2.0/09 /APIA/VIAA/044.

References

1. Arnicane, V.: Use of Non-IT Testers in Software Development. In Münch, J., Abrahamsson P. (eds) 8th International Conference on Product Focused Software Process Improvement (PROFES 2007), LNCS, vol. 4589, pp. 175--187, Berlin, Springer (2007)
2. Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M., Wiedenbeck, S.: The state of the art in end-user software engineering. ACM Comput. Surv. 43, 3, 21:1--21:44 (2011)
3. Lieberman, H., Paterno, F., Wulf, V., (Eds.): End-User Development. Kluwer/ Springer (2006)
4. Yoder, A.G., Cohn, D.L.: Domain-specific and general-purpose aspects of spreadsheet languages. Distributed Computing Research Lab, University of Notre Dame. http://www-sal.cs. uiuc.edu/~kamin/dsl/papers/yoder.ps. (2002)
5. Myers, B., Ko, A.J., Burnett, M.M.: Invited research overview: end-user programming. In CHI '06 extended abstracts on Human factors in computing systems (CHI EA '06), pp. 75--80. ACM, New York (2006)
6. Baron, M., Girard, P.: Bringing Robustness to End-User Programming. In Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01) (HCC '01), pp. 142—149, IEEE Computer Society, Washington, DC (2001)
7. Obrenovic, Z.; Gasevic, D.: End-User Service Computing: Spreadsheets as a Service Composition Tool. Services Computing, IEEE Transactions on , vol.1, no.4, pp. 229--242 (2008)

8. Dragan, M.: Using Excel as a front-end for MLF. Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2005. Seventh International Symposium on , pp. 114--117 (2005)
9. Erwig, M.: Software Engineering for Spreadsheets. Software, IEEE , vol.26, no.5, pp. 25--30 (2009)
10. Erwig, M., Abraham, R., Cooperstein, I., Kollmansberger, S.: Automatic generation and maintenance of correct spreadsheets. In Proceedings of the 27th international conference on Software engineering (ICSE '05). pp. 136—145, ACM, New York (2005)
11. Encyclopedia of Database Systems, Liu L., Özsu M. T. (eds.), Springer-Verlag (2009)