Science of Computer Programming 96 (2014) 254–275

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Model-based programming environments for spreadsheets

Jácome Cunha [a,c], Jorge Mendes [a,c], João Saraiva [a], Joost Visser [b]

[a] HASLab/INESC TEC, Universidade do Minho, Portugal
[b] Software Improvement Group & Radboud University Nijmegen, The Netherlands
[c] CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal

HIGHLIGHTS

- We infer a relational model from spreadsheet data.
- The extracted model is embedded into the spreadsheet.
- We develop a model-based spreadsheet programming environment.
- The environment provides advanced editing assistance.
- We validate our techniques via an empirical study.

ARTICLE INFO

Article history:
Received 25 March 2013
Received in revised form 28 January 2014
Accepted 2 February 2014
Available online 12 February 2014

Keywords:
Spreadsheets
Model-driven engineering
Model-driven spreadsheets
Empirical validation

ABSTRACT

Spreadsheets can be seen as a flexible programming environment. However, they lack some of the concepts of regular programming languages, such as structured data types. This can lead the user to edit the spreadsheet in a wrong way and perhaps cause corrupt or redundant data.

We devised a method for extraction of a relational model from a spreadsheet and the subsequent embedding of the model back into the spreadsheet to create a model-based spreadsheet programming environment. The extraction algorithm is specific for spreadsheets since it considers particularities such as layout and column arrangement. The extracted model is used to generate formulas and visual elements that are then embedded in the spreadsheet helping the user to edit data in a correct way.

We present preliminary experimental results from applying our approach to a sample of spreadsheets from the EUSES Spreadsheet Corpus.

Finally, we conduct the first systematic empirical study to assess the effectiveness and efficiency of this approach. A set of spreadsheet end users worked with two different model-based spreadsheets, and we present and analyze here the results achieved.

1. Introduction

Developments in programming languages are changing the way in which we construct programs: naive text editors are now replaced by powerful programming language environments which are specialized for the programming language under consideration and which help the user throughout the editing process. Helpful features like highlighting keywords of the language or maintaining a beautified indentation of the program being edited are now provided by several text editors. Recent advances in programing languages extend such naive editors to powerful language-based environments [1–6].

E-mail addresses: jacome@di.uminho.pt (J. Cunha), jorgemendes@di.uminho.pt (J. Mendes), jas@di.uminho.pt (J. Saraiva), j.visser@sig.eu (J. Visser).

http://dx.doi.org/10.1016/j.scico.2014.02.002
0167-6423/

Language-based environments use *knowledge* of the programming language to provide the users with more powerful mechanisms to develop their programs. This knowledge is based on the *structure* and the *meaning* of the language. To be more precise, it is based on the syntactic and (static) semantic characteristics of the language. Having this knowledge about a language, the language-based environment is not only able to highlight keywords and beautify programs but it can also detect features of the programs being edited that, for example, violate the properties of the underlying language. Furthermore, a language-based environment may also give information to the user about properties of the program under consideration. Consequently, language-based environments guide the user in writing correct and more reliable programs.

Spreadsheet systems can be viewed as programming environments for non-professional programmers, the so-called *end-user* programmers. In order to improve end-user productivity, modern spreadsheet systems offer some of the mechanisms found in programming environments, as, for instance, auto-completion of names/cells, or warnings when mismatching data types. These mechanisms, however, are very limited: they focus on the *structure* of the data, and not on the *meaning* of the spreadsheet data. As a consequence, the end-user guidance provided by spreadsheets systems is very weak when compared to modern programming environments.

This paper extends previous work on generating powerful spreadsheet environment [7,8]: we propose a technique to enhance a spreadsheet system with mechanisms to guide end users to introduce correct data. A background process adds formulas and visual objects to an existing spreadsheet, based on a relational database schema. To obtain this schema, or model, we follow the approach used in language-based environments: we use the *knowledge* about the data already existing in the spreadsheet to guide end users in introducing correct data. The knowledge about the spreadsheet under consideration is based on the *meaning* of its data that we infer using data mining and database normalization techniques.

Data mining techniques specific to spreadsheets are used to infer *functional dependencies* from the spreadsheet data. These functional dependencies define how certain spreadsheet columns determine the values of other columns. Database normalization techniques, namely the use of normal forms [9], are used to eliminate redundant functional dependencies, and to define a relational database model. Knowing the relational database model induced by the spreadsheet data, we construct a new spreadsheet environment that not only contains the data of the original one but also includes advanced features which provide information to the end user about correct data that can be introduced. We consider several types of advanced features: *auto-completion of column values*, *non-editable columns*, *safe deletion of rows*, *key columns* and *reference columns*.

Besides presenting in this paper an extension of our previous work on generating spreadsheet programming environments, we also include in this paper both the first experimental results validating the proposed techniques, which are obtained by considering a large set of spreadsheets included in the EUSES Spreadsheet Corpus [10], and an empirical study evaluating the effectiveness and efficiency of the model-based spreadsheets. Both experiments show that our techniques work not only for database-like spreadsheets, like the example we will use throughout the paper, but they work also for realistic spreadsheets defined in other contexts (for example, inventory, grades or modeling).

This paper is organized as follows. Section 2 presents an example used throughout the paper. Section 3 presents our algorithm to infer functional dependencies and how to construct a relational model. Section 4 discusses how to embed assisted editing features into spreadsheets. A preliminary evaluation of our techniques is present in Section 5. In Section 6 we present an empirical validation of the spreadsheets we generate. Section 7 discusses related work and Section 8 concludes the paper.

2. Spreadsheet programming environments

In order to present our approach we shall consider the following example taken from [11] and modeled in a spreadsheet as shown in Fig. 1.

	A	B	C	D	E	F	G	H	I	J	K	L
1	clientNr	propNr	cName	pAddress	country	rentStart	rentFinish	days	rent	total	ownerNr	oName
2	cr76	pg4	john	6 Lawrence	UK	01/07/00	08/31/01	602	50	30100	co40	tina
3	cr76	pg16	john	5 Novar Dr.	UK	09/01/01	09/01/02	365	70	25550	co93	tony
4	cr56	pg4	aline	6 Lawrence	UK	09/02/99	06/10/00	282	50	14100	co40	tina
5	cr56	pg36	aline	2 Manor Rd	UK	10/10/00	12/01/01	417	60	25020	co93	tony
6	cr56	pg16	aline	5 Novar Dr.	UK	11/01/02	08/10/04	648	70	45360	co93	tony

Fig. 1. A spreadsheet representing a property rental system.

This spreadsheet contains information related to a housing rental system. It gathers information about clients, owners, properties, prices and rental periods. The name of each column gives a clear idea of the information it represents. We extend this example with three additional columns, named *days* (that computes the total number of rental days by subtracting the column *rentStart* to *rentFinish*), *total* (that multiplies the number of rental days by the rent per day value, *rent*) and *country* (that represents the property's country). As usually in spreadsheets, the columns *days* and *rent* are expressed by formulas.

This spreadsheet defines a valid model to represent the information of the rental system. However, it contains redundant information: the displayed data specifies the house rental of two clients (and owners) only, but their names are included five times, for example. This kind of redundancy makes the maintenance and update of the spreadsheet complex and error-prone. A mistake is easily made, for example, by mistyping a name, thus corrupting the data on the spreadsheet.