Towards a DSL for Educational Data Mining

Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez(✉)

Dpto. Ingeniería Informática y Electrónica,
Universidad de Cantabria, Santander, Spain
{alfonso.delavega,diego.garcia,marta.zorrilla,p.sanchez}@unican.es

Abstract. Nowadays, most companies and organizations rely on computer systems to run their work processes. Therefore, the analysis of how these systems are used can be an important source of information to improve these work processes. In the era of Big Data, this is perfectly feasible with current state-of-art data analysis tools. Nevertheless, these data analysis tools cannot be used by general users, as they require a deep and sound knowledge of the algorithms and techniques they implement. In other areas of computer science, domain-specific languages have been created to abstract users from low level details of complex technologies. Therefore, we believe the same solution could be applied for data analysis tools. This article explores this hypothesis by creating a Domain-Specific Language (DSL) for the educational domain.

Keywords: Domain-specific languages · Big data · Educational data mining

1 Introduction

Nowadays, most work processes in companies and organizations are supported by software systems. Thus, the way in which people interact with these systems reflects somehow how these processes are actually executed. Therefore, a careful analysis of this interaction can help to find out flaws of these processes that might be removed [4].

For instance, let us suppose a company which wants to reduce the number of products that are returned after having been shipped. In this scenario, the company managers might be interested in getting answers for questions such as: *"What features share those products that are returned by customers?"*; or *"What is the profile of the unsatisfied customers?"*. Decision makers need to know these answers before adopting corrective actions.

Currently, it is feasible to perform this data analysis by using Big Data technologies [3]. For instance, the profiles of the unsatisfied customers can be computed using clustering techniques [8]. These techniques require a sound knowledge of the algorithms and mathematical foundations they use. Nevertheless, average decisions makers do not have this knowledge.

For example, to execute a clustering, the user should know how a clustering algorithm like *K-means* [1] works, how its parameters must be configured,

J.-L. Sierra-Rodríguez et al. (Eds.): SLATE 2015, CCIS 563, pp. 79–90, 2015.
DOI: 10.1007/978-3-319-27653-3_8

or what are the advantages of *K-means* as compared to other clustering algorithms, such as *X-means* [13]. Since decision makers lack of this knowledge, they need to rely on third-parties to carry out these data analysis processes, which leads to a costs increment and a productivity reduction.

In other areas of software development, *Domain Specific Languages (DSLs)* [9,18] have been created in order to allow users without expertise in a certain technology to use it. This is achieved by abstracting low level details of the underlying technology and by using a syntax and a terminology familiar to the end-user.

Therefore, we propose to build DSLs for data analysis. These DSLs would allow decision makers to formulate queries about the performance of a business process using a syntax and terminology familiar to them. Then, these queries would be automatically transformed into invocations of specific algorithms for data analysis. The DSL syntax should hide all the details associated with data analysis techniques to the end-user, who might remain unaware of how these techniques are used.

This article explores the feasibility of this idea by showing how a DSL with these characteristics can be developed for the e-learning domain. The objective of this DSL is to analyze the performance of a course hosted on an e-learning platform, such as *Moodle*, by using data, like the students' activity, gathered via this kind of platform. The final users of the DSL will be teachers and instructors, so it must use a syntax and a terminology familiar to them. Similar DSLs might be created for other domains following the process described in this article.

For the development of the DSL, we will make use of modern model-driven engineering techniques. More specifically, we will follow the development process proposed by Kleppe [9].

After this introduction, this article is structured as follows: Sect. 2 describes the domain our DSL targets. Section 3 comments on related work. Sections 4 and 5 explain how a DSL for the educational domain has been developed. Finally, Sect. 6 discusses the benefits of this work and concludes this article.

2 Educational Data Mining

The first step to develop a DSL, according to [12], is to know for what purpose the DSL will be used and obtain a sound knowledge of the domain it will target. In our case, this domain will be the educational domain, and we are mainly interested in knowing for what kind of questions decision makers would like to get an answer. Moreover, we are also interested in discovering what data is available to compute these answers. The domain information has been obtained using our own experience in the educational data mining domain as well as the assistance of several external teachers and instructors. In the following, the Educational Data Mining domain is described briefly.

Data Mining is the process of discovering interesting patterns and knowledge from large amounts of data [8]. In the last few years, it has been applied to the educational domain, what is known as *Educational Data Mining (EDM)* [17]. Educational Data Mining aims to take advantage of the data gathered by e-learning platforms, such as *BlackBoard* [16] or *Moodle* [15], which store data related to the activity carried out by the students of their courses. Educational Data Mining is defined as *"an emerging discipline, concerned with developing methods for exploring the unique types of data that come from educational settings, and using those methods to better understand students, and the settings which they learn in."* [17]. The discovered information could be useful for teachers and instructors in order to improve the performance of their teaching-learning processes.

For instance, at the beginning of a course, a teacher might be interested in what kind of students' profiles exist. Based on the obtained information, the teacher might adapt the course before it starts in order to tune it for these students. Thus, at the beginning of the course, the teacher could ask: *"What are the profiles of my students?"*. This information can be computed by using *clustering techniques* [8] on the students' demographic and activity data.

When the course finishes, teachers are usually worried about the students that have not passed the course. Therefore, they would like to refine the previous question and ask: *"What are the profiles of the students who have not passed?"*. As before, this information can be computed using clustering techniques on the students' data, but removing those students that have passed from this dataset. Moreover, teachers are obviously interested in asking *"What are the reasons why my students failed?"*. This might be partially answered by applying *classification techniques* [8] on the students' data, by analysing the student activity logs to find out these reasons.

Obviously, most teachers know nothing about clustering and classification rules, so they cannot use these techniques directly by themselves. This is the reason behind the aim of hiding these details to the end-user.

Next section analyses whether this objective can be achieved using current state-of-art techniques.

3 Related Work

To the best of our knowledge, there is little work done about how to make data analysis techniques more usable by decision makers. The approaches which tackle this issue can be grouped in two sets.

The first group aims to assist decision makers in the process of defining a data analysis process. For instance, [5] defines a method where end-users are prompted with different questions, which guide them in the definition of a data mining process that fits in with their needs.

For example, a question could be if the decision maker is interested in computing the profiles of a certain dataset. If so, the user is asked for more detailed information that is required to execute this task. Some of these questions might result confusing. As an example, the user should be able to answer about how a certain data is represented, if as a string label or as a numerical value. An average decision maker might not know these technical details. Moreover, answering these questions can be a large and tedious process, which could lead to build wrong mining models or to stop using the tool.

In [2], a query-by-example based language is defined. In basic query-by-example, the decision maker constructs a prototype of an answer for the question he or she would like to ask. This prototype is a table, where each column represents an attribute of the desired answer. These columns can be constrained to certain values, which are used to select the desired results. The system depicted in [2] enhances this table with specific columns to execute data mining processes. Again, the information we need to supply in these columns requires some knowledge of the underlying data analysis technique to be applied, so the user is not completely unaware of these techniques. Moreover, the construction of these prototypes is based on data warehouse concepts, such as OLAP (*On-Line Analytical Processing*) [20]. Average decision makers also often lack of this kind of knowledge.

In the second group, there are software applications with prebuilt data mining processes which can be directly executed by decision makers. An example of this strategy is *E-learning Web-Miner (ElWM)* [21]. ElWM is a web-based application whose objective is to allow instructors to analyse the performance of a course hosted in an e-learning platform. At the time of writing this article, ElWM offers instructors answers to three different queries: (1) what kinds of resources are frequently used together (e.g., forum, mail) in each learning session; (2) what are the profiles of the different sessions carried out by students; and (3) what are the profiles of the students enrolled in a course.

In this case, the main limitation is that the set of queries is fixed and they cannot be refined without modifying the application. For instance, if we wanted to compute the profiles of assignments which students have failed; or the profiles of students that do not pass the course, we would need to update the application to allow this more specific filtering.

By developing DSLs for data analysis, we expect to overcome these shortcomings. Next sections describe how this task is accomplished for the educational domain.

4 Grammar Specification

As previously commented, we will follow the process proposed by Kleppe [9] for the development of the DSL. According to this process, the first step to implement a DSL, once the knowledge about the target domain has been collected, is to specify its grammar. Next subsections describe how this step is accomplished.

The definition of a grammar for a DSL, following a model-driven perspective, implies the definition of an abstract syntax and a concrete syntax. The abstract syntax specifies the grammar of a language independently of how this model is represented. The concrete syntax is a specific rendering, either textual or visual, for the abstract syntax. We describe both elements below.

4.1 Abstract Syntax

Abstract syntaxes are usually specified by using *metamodels* [11]. A metamodel can be considered as a model of the syntax of a language. For the construction of

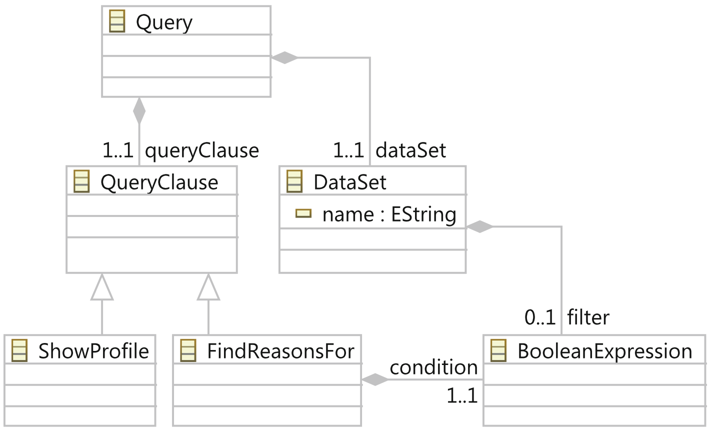

Fig. 1. Abstract syntax of our DSL

this metamodel, we have used Ecore [19], which is the de-facto standard language for metamodeling. Figure 1 shows the metamodel for our DSL syntax.

According to this metamodel, our language allows us to write queries. A *Query* has a *QueryClause*. In the figure, two query clauses are depicted: *ShowProfile* and *FindReasonsFor*. A query clause can be viewed as a command that hides a data mining technique. Moreover, each query has an associated *DataSet*, which must be available in a well-defined location.

Moreover, a data source can have an associated *filter*. A *filter* is a boolean expression that selects the subset of instances of a data source which satisfies such expression. The abstract syntax for boolean expressions are not shown in this article for the sake of simplicity and brevity, as this syntax is probably known by the reader.

Filters are used to apply a query clause to a specific subset of a data source. For instance, an instructor might be interested in selecting students that: (1) do not pass a course; (2) drop out; or (3) are above or below a certain age, among other options. Obviously, these filters must be written using the attributes of the data source. For instance, if students' age is not stored in the database, it could not be used in a filter.

In the case of the *FindReasonsFor* clause, an additional *condition* is required because the goal of this query is to compute the reason why certain instances of the data source satisfy a certain condition. As before, this condition is a boolean expression.

After developing the abstract syntax of our DSL, the next step is to specify its concrete syntax, which is described in the following subsection.

4.2 Concrete Syntax

We have opted for developing a textual syntax for our DSL. However, this issue needs to be further investigated, as some decision makers might prefer a graphical notation.

```
00 grammar es.unican.dslEdm.Dsl
01         with org.eclipse.xtext.common.Terminals

02 import "EdDataMiningMetamodel"
03 import "http://www.eclipse.org/emf/2002/Ecore" as ecore

04 Query returns Query:
05  queryClause=QueryClause 'of' dataSet=DataSet;

06 QueryClause returns QueryClause:
07   ShowProfile | FindReasonsFor;

08 ShowProfile returns ShowProfile:
09   'show_profile';

10 FindReasonsFor returns FindReasonsFor:
11   'find_reasons_for' condition=BooleanExpression;

12 DataSet returns DataSet:
13   name=ID ('with' filter=BooleanExpression)?;

...
```

Fig. 2. Textual concrete syntax for our DSL

For the definition of the textual concrete syntax, we have used *Xtext* [6], which allows DSL developers to define a textual syntax from a Ecore metamodel. Using *Xtext*, a grammar is defined following a notation similar to EBNF (*Extended Backus–Naur Form*), but where the production rules are enhanced with constructions to create instances of metaclasses from the metamodel as the grammar is parsed.

Figure 2 shows the concrete syntax for our DSL. Lines 00–03 specify: (1) the namespace and name for the grammar; (2) include a convenience package called *Terminals*, provided by *Xtext*; and, (3) specifies that the grammar will be based on the Ecore and the *EdDataMiningMetamodel* metamodel, which corresponds to the abstract syntax depicted in Fig. 1.

Then, Lines 04–05 specify *Query* as the entry point of our grammar. A *Query* is composed of a *QueryClause*, followed by the *of* keyword and the specification of a *Dataset*. Both lines are equivalent to the EBNF rule *Query* :: = *QueryClause* *"of" DataSet*.

Moreover, in Line 04, the *return Query* clause specifies that an instance of the *Query* metaclass (see Fig. 1) must be created when this production rule is executed. Furthermore, the results of executing the *DataSet* and *QueryClause* production rules must be assigned to the *queryClause* and *dataSet* attributes of the *Query* metaclass, respectively.

Similarly, a *QueryClause* can be either a *ShowProfile* or a *FindReasonsFor* (Lines 06–07) clause. In the first case, the query clause is simply written using the *show_profile* keyword (Lines 08–09). In the second case, after the keyword *find_reasons_for*, a boolean expression that serves as *condition* for evaluating the query is required (Lines 10–11).

```
show_profile of Students;

show_profile of Students with courseOutcome=fail;

find_reasons_for courseOutcome=fail of Students;
```

Fig. 3. Queries written using the DSL

Finally, a *Dataset* is simply represented by an identifier plus an optional filter definition (Lines 12–13). This identifier must correspond to an available dataset. This constraint is checked by means of external rules.

Once the grammar has been specified, a full editor for our grammar, with syntax colouring, helpers and automatic formatting, as well as parsing, type-checking and validation capabilities can be automatically generated by *Xtext*. Using this editor, queries as shown in Fig. 3 can be written.

Thus, instructors can now write queries to analyse course performance by using a terminology that is familiar to them. The next step is to provide execution capabilities to these queries, which is achieved by translating them into Java code.

5 Query Execution

To compute the result of a query, data mining techniques are used. For instance, to identify profiles in a dataset, clustering techniques must be chosen. Therefore, the strategy to execute a query is to transform it into a Java code snippet which invokes a prebuilt implementation of the corresponding data mining algorithm. In our case, these prebuilt implementations are provided by *Weka* [7], a widely used data mining tool suite.

It should be taken into account that most data mining algorithms require the specification of a set of input parameters, which are necessary for tuning the algorithm. For instance, most clustering techniques require the specification of the number of clusters to be built. Obviously, if the ultimate goal of the DSL is to abstract the end user from data mining techniques, it cannot be expected that the end-user provides the values for these parameters.

Therefore, these parameters have to be self-computed. Currently, there is a research area inside the data mining field, known as *parameter-less data mining*, that aims to build self-configuring data-mining algorithms. Thus, these techniques will be used whenever possible.

To illustrate how this code generation process works, we describe how the *Show Profile* queries are transformed into Java code. The query *show_profile of Students with courseOutcome=fail;* will be used as an example.

The code generation process has been implemented using templates. More specifically, we have used EGL (*Epsilon Generation Language*) [10], which is a language that allows code generation from Ecore-based models.

```
00  DataSource source = new DataSource("[%=query.dataSet.name%].arff");
01  Instances ins = source.getDataSet();
02  Instances insFiltered = ins;
    [%
03  if (query.dataSet.filter.isDefined()) {
04    var filterConstraints = query.dataSet.filter.operations;
05    for (constraint in filterConstraints) { %]
06          [%=constraint.toFilter()%]
07  [%}
08  }
    %]
09  XMeans xm = new XMeans();
10  xm.setMinNumClusters(Clustering.XMEANS_MIN_NUM_CLUSTERS);
11  xm.setMaxNumClusters(Clustering.XMEANS_MAX_NUM_CLUSTERS);
12  xm.buildClusterer(insFiltered);
13  Visualizer.saveClusterer(xm, Visualizer.ClusterType.XMEANS);
```

Fig. 4. General clustering template

```
00 [%@template
01   operation EqualityComparison toFilter() {
02     var attrName = "attr" + self.attrName;
03     var valueName = "value" + self.attrName; %]
04     Attribute [%=attrName%] = ins.attribute("[%=self.attrName%]");
05     String [%=valueName%] = "[%=self.value%]";
06     [% var filterName = "filter" + self.attrName; %]
07     RemoveWithValues [%=filterName%] = new RemoveWithValues();
08     [%=filterName%].setAttributeIndex(
09         Integer.toString([%=attrName%].index() + 1));
10     [%=filterName%].setNominalIndices(
11         Integer.toString([%=attrName%].indexOfValue([%=valueName%]) + 1));
12     [%=filterName%].setInputFormat(ins);
13     [%=filterName%].setInvertSelection(true); // matching entries
14     insFiltered = Filter.useFilter(insFiltered, [%=filterName%]);
15 [%} %]
```

Fig. 5. Filter for equality comparisons

As a first step, a template is selected based on the query clause. A fragment of the template applied for the *Show Profile* queries can be seen in Fig. 4, whereas the code generated by this template for our example query is shown in Fig. 6. The code generation process based on this template works as follows:

1. First, the dataset to be analysed is loaded, using the corresponding Weka helper classes. In this case, Lines 00–01 of Fig. 4 are in charge of generating Lines 04–05 of Fig. 6. The name of the dataset is obtained from the attribute name of the *Dataset* metaclass (Fig. 4, Line 00). As previously indicated, the parser must have checked that a dataset with that name exists.
2. In case the dataset has an associated filter, the code to perform this filtering must be generated. Consequently, the boolean expression which defines the filter must be transformed into the corresponding Weka code to filter a dataset according to the values of certain attributes. Figure 4, Lines 03–08

```
00 package processes;
01 import weka.[...]

02 public class ClusteringSnippet {
03   public static void main(String[] args) throws Exception {

04     DataSource source = new DataSource("Students.arff");
05     Instances ins = source.getDataSet();

06     Instances insFiltered = ins;
07     Attribute attrcourseOutcome = ins.attribute("courseOutcome");
08     String valuecourseOutcome = "fail";
09     RemoveWithValues filtercourseOutcome = new RemoveWithValues();
10     filtercourseOutcome.setAttributeIndex(Integer
               .toString(attrcourseOutcome.index() + 1));
11     filtercourseOutcome
               .setNominalIndices(Integer.toString(attrcourseOutcome
                       .indexOfValue(valuecourseOutcome) + 1));
12     filtercourseOutcome.setInputFormat(ins);
13     filtercourseOutcome.setInvertSelection(true); // matching entries
14     insFiltered = Filter.useFilter(insFiltered, filtercourseOutcome);

15     XMeans xm = new XMeans();
16     xm.setMinNumClusters(Clustering.XMEANS_MIN_NUM_CLUSTERS);
17     xm.setMaxNumClusters(Clustering.XMEANS_MAX_NUM_CLUSTERS);
18     xm.buildClusterer(insFiltered);
19     Visualizer.saveClusterer(xm, Visualizer.ClusterType.XMEANS);
    }
  }
```

Fig. 6. Code generated after processing the example query

show the template code which processes the constraints of the boolean expression and converts them to Java code. For each constraint, the *toFilter* method is invoked. The implementation of this method is different for each kind of constraint. In our example, an equality comparison is used to filter those students whose *courseOutcome* is *fail*. Therefore, the *toFilter* method corresponding to equality comparisons is invoked. The code generation template corresponding to this operation is shown in Fig. 5. This template makes use of the existent Weka filter *RemoveWithValues* to achieve this goal. Figure 6, Lines 07–14 show the resulting code for our concrete example.

3. Then, the code for executing the clustering algorithm on the loaded dataset is generated (Fig. 4, Lines 09–12, Fig. 6, Lines 15–18). In our case, the *X-means* algorithm [13] is selected to perform the clustering. The advantage of this algorithm is that it can estimate the number of clusters that should be created for a particular dataset. *X-means* only requires that this number is bound to a certain range. Therefore, if the lower and upper bounds of this range are set to proper values, the *X-means* algorithm can be used as a self-configuring algorithm. Since teachers expect to find at least two different students groups, 2 is a reasonable lower bound in this case. For a normal course, 20 is a number of clusters high enough to be considered as infinite, thus it is a reasonable upper bound for our algorithm. Therefore, the responsibility of determining

the number of clusters to be created is delegated into the *X-means* algorithm, which automatically calculates it.

4. Finally, Fig. 6, Line 13 shows how the result of the *X-means* algorithm is placed in an output file, which is read by a visualization tool in order to adequately render the results in a user-friendly way.

The code generation process for the *FindReasonsFor* queries would be similar, but in this case, a classification algorithm would be invoked, precisely, the *J48* implementation provided by Weka of the *C4.5* decision tree algorithm [14]. Therefore, a different EGL template would be used in this case.

With this last step, the development of our DSL for Educational Data Mining is finished. DSLs for applying data mining techniques in other domains might be developed following a similar process. Several excerpts of the DSL are agnostic of the target domain, thus they could be reused and, consequently, a remarkable reduction in the cost and development effort of a new DSL could be achieved.

Next section discusses whether this DSL satisfies the objectives of this work and concludes this article.

6 Conclusions

This article has shown how a Domain-Specific Language for Educational Data Mining can be developed. This DSL allows teacher and instructors of courses hosted in e-learning platforms to analyse the performance of their teaching-learning processes by means of applying data mining techniques on the data contained in such a platform. The DSL approach provides two benefits as compared to current state-of-art techniques.

First, the DSL abstracts low level-details of data analysis techniques, so it can be used by instructors without any knowledge of these techniques. Thus, our approach offers a solution to bridge the gap between data analysis tools and decision makers. The DSL syntax only contains high-level keywords and references to entities and attributes of the target domain data model. Thus, the DSL contains a terminology that should be known by the decision makers, who would be instructors and teachers in the case of the educational domain.

Secondly, the DSL is flexible enough to support the elaboration of arbitrary complex new queries. This is an advantage as compared to approaches that develop tools able to compute concrete tasks. ElWM [21] is an example of such a tool. As commented in Sect. 3, using ElWM students' profiles of a course can be computed. However, the profiles of students above a certain age cannot be computed without modifying the application. Similarly, we cannot calculate profiles of other entities, such as assignments, without updating the tool. This is, each time we want to modify a query, the application must be updated to support it.

Oppositely, the DSL offers a more flexible interaction. There exists limitations, as decision makers cannot ask any arbitrary question and they must adhere to the available query clauses. Thus, the included set of clauses should cover the potential questions decision makers are interested in asking. Moreover, the queries must be written following the syntactic rules of the grammar, as the necessity to parse them with a computer prevents the usage of most informal natural language expressions.

As future work, we expect to add more query options to the DSL for the educational domain, as well as to develop DSLs for other domains. More specifically, we are interested in developing DSLs for the performance analysis of work processes in the public administration.

Acknowledgements. This work has been partially funded by the Government of Cantabria (Spain) under the doctoral studentship program from the University of Cantabria, and the Spanish Government and FEDER funder under grant TIN2011-28567-C03-02 (HI-PARTES).

References

1. Arthur, D., Vassilvitskii, S.: K-means++: the advantages of careful seeding. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1027–1035, New Orleans (Louisiana, USA), January 2007
2. Azevedo, A., Santos, M.: Binding data mining to final business users of business intelligence systems. In: 1st International Conference on Intelligent Systems and Applications (Intelli), pp. 7–12, April–May 2012
3. Baesens, B.: Analytics in a Big Data World: The Essential Guide to Data Science and Its Application. Wiley, New York (2014)
4. Bughin, J., Chui, M., Manyika, J.: Clouds, big data and smart assets: ten tech-enabled business trendsto watch. McKinsey Q. **56**, 1–14 (2010)
5. Espinosa, R., García-Saiz, D., Zorrilla, M., Zubcoff, J.J., Mazón, J.-N.: Enabling non-expert users to apply data mining for bridging the big data divide. In: Ceravolo, P., Accorsi, R., Cudre-Mauroux, P. (eds.) SIMPDA 2013. LNBIP, vol. 203, pp. 65–86. Springer, Heidelberg (2015)
6. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: Companion to the 25th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA), pp. 307–309, Reno/Tahoe (Nevada, USA), October 2010
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
8. Han, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann, USA (2005)
9. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages using Metamodels. Addison-Wesley Professional, Reading (2008)
10. Kolovos, D.S., Paige, R.F., Rose, L.M., Williams, J.: Integrated model management with epsilon. In: France, R.B., Kuester, J.M., Bordbar, B., Paige, R.F. (eds.) ECMFA 2011. LNCS, vol. 6698, pp. 391–392. Springer, Heidelberg (2011)
11. Kühne, T.: Matters of (meta-)modeling. Softw. Syst. Model. **5**(4), 369–385 (2006)
12. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
13. Pelleg, D., Moore, A.: X-means: extending K-means with efficient estimation of the number of clusters. In: Proceedings of the 17th International Conference on Machine Learning, pp. 727–734. Morgan Kaufmann (2000)

14. Quinla, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco (1993)
15. Rice, W.: Moodle E-Learning Course Development. Packt Publishing, Birmingham (2006)
16. Rice, W.: Blackboard Essentials for Teachers. Packt Publishing, Birmingham (2012)
17. Romero, C., Ventura, S.: Data mining in education. Wiley Interdisc. Rev.: Data Mining Knowl. Discov. **3**(1), 12–27 (2013)
18. Sierra, J.L.: Language-driven software development (invited talk). In: Pereira, M.J.V., Leal, J.P., Simões, A. (eds.) 3rd Symposium on Languages, Applications and Technologies. OpenAccess Series in Informatics (OASIcs), vol. 38, pp. 3–12 (2014)
19. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, vol. 2. Addison-Wesley Professional, Reading (2008)
20. Wrembel, R., Koncilia, C.: Data Warehouses and Olap: Concepts, Architectures and Solutions. IRM Press, London (2006)
21. Zorrilla, M., García-Saiz, D.: A service-oriented architecture to provide data mining services for non-expert data miners. Decis. Support Syst. **55**(1), 399–411 (2013)