FLANDM: a development framework of domain-specific languages for data mining democratisation

Alfonso de la Vega*, Diego García-Saiz, Marta Zorrilla, Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica, Universidad de Cantabria, Avda. de los Castros s/n, Santander 39005, Spain

Keywords:
Model-driven software development
Domain-specific languages
Data mining
Data mining democratisation

A B S T R A C T

Companies have an increasing interest in employing data mining to take advantage of the vast amounts of data their systems store nowadays. This interest confronts two problems: (1) business experts usually lack the skills required to apply data mining techniques, and (2) the specialists who know how to use these techniques are a scarce and valuable asset. To help democratise data mining, we proposed, in a previous work, the development of domain-specific languages (DSLs) that hide the complexity of data mining techniques. The objective of these DSLs is to allow business experts to specify analysis processes by using high-level primitives and terminology from the application domain. These specifications would then be automatically transformed into a low-level, executable form. Although these DSLs might offer a promising solution to the aforementioned problems, their development from scratch requires a considerable effort and, consequently, they are costly. In order to make these languages affordable, we present FLANDM, an ecosystem devised for the rapid development of DSLs for data mining democratisation. FLANDM provides a base infrastructure that can be easily customised for the particularities of each domain, enabling controlled and systematic reuse of previously developed artefacts. By using FLANDM, new DSLs for data mining democratisation can be defined achieving a 50% of reduction in their development costs.

1. Introduction

Data mining techniques are nowadays having an ever-increasing popularity. The more services and processes are digitalized in our society, the more data about them are stored. These data, when properly analysed, can be used to make better and more informed decisions [1]. For instance, *Netflix* determines what media contents to produce based on the analysis of their users activity and ratings information [2].

Unfortunately, the skills required to define and execute data mining processes are very specific, including advanced statistics, specialised algorithms and data management techniques [3]. These requirements make very difficult for business managers, who are experts in their field but not in analysis techniques, to analyse bundles of data by themselves. Therefore, the commonly applied solution involves hiring a specialist, who defines these data mining processes on behalf of the business managers. This fact has increased the demand of these specialists, currently known as *data scientists* [4]. Consequently, data scientists have become a scarce and expensive asset [5].

To alleviate this problem, in a previous work, we proposed to define Domain-Specific Languages (DSLs) that contribute to Data Mining Democratisation [6]. In the following, we refer to these languages as DMDLs, from Data Mining Democratisation Languages. The objective was that business experts could use a language, comprised of a set of high-level primitives and terms related to their domain, to easily specify and execute data mining tasks. This language should hide the complexity of data mining techniques to business experts as much as possible. The feasibility of this idea was evaluated by developing a DMDL for the educational domain [7]. Using this DMDL, teachers were able to study the performance of their courses by analysing the data stored in their e-learning platforms.

Although our previous work demonstrated that DMDLs might contribute to data mining democratisation, we noticed that the development of these languages required an important effort and, consequently, they were costly. It should be remarked that each domain might require its own DMDL in order to capture its peculiarities, so the development of a generic, domain-independent language was not a solution. Nevertheless, these DMDLs shared several features, so a significant part of their elements could be reused among domains. This reuse should help reduce development costs.

This work explores the previous idea and presents FLANDM (*development Framework of LANguages for Data Mining*), an infrastructure for the rapid development of domain-specific languages for data mining democratisation. It provides a set of generic and reusable assets that can be easily and quickly customised to adapt them to a new domain. These assets conform a generic DMDL structure, which is extended by the definition of concrete new DMDLs [8]. These new DMDLs could be seen as internal DSLs of FLANDM [9]. This way, the development effort for the creation of each new DMDL is decreased. Moreover, due to the highly modular structure of FLANDM, designed to facilitate the reuse of their components, DMDLs developed with it can better deal with changes, which decreases their maintenance cost. Therefore, the contribution of FLAMDM is twofold, as it decreases both development and maintenance cost of a DMDL.

A preliminar version of this work was presented at the 7*th* International Conference on Model and Data Engineering (MEDI 2017) [10]. This article extends that work by providing a comprehensive evaluation on how far its research goals were achieved. Moreover, some concepts have been explained more in-depth, and further examples are provided.

FLANDM was evaluated by creating four languages for different case studies and domains [11]. Due to the use of FLANDM, about 55% of the DMDLs' final structure came from reused components. This reuse accomplished an average reduction of development costs of these DMDLs by 50% when compared with the cost of developing them from scratch. With respect to maintenance ease, the modular structure of FLANDM provided a good separation of concerns into simple components. At first, the modularity increased the number of affected components by the introduced changes when compared with a more monolithic structure. However, as these components were simpler than the ones of the monolithic version, the overall effort of applying the changes was reduced.

The rest of this paper is organized as follows: Section 2 introduces to the field of data mining for non-expert users through a literature review. In Section 3, the motivation behind this work is explained. Next, the structure of FLANDM is detailed in Section 4. Section 5 evaluates our approach. Finally, Section 6 summarizes this paper and comments on future work.

2. Related work: data mining for non-expert users

As starting point of our research, we checked the following hypothesis: state-of-the-art data mining techniques cannot be used by people without a solid expertise on them. This hypothesis was verified by performing a systematic literature review, whose objective was to analyse works that aimed at making data mining techniques attainable to non-expert users. These works are often framed into the concept of *data mining democratisation* [6].

The review was performed[1] following the guidelines of Kitchenham et al [12] and Wohlin et al [13]. During this review, more than 500 articles and tools were initially considered. The tools and research work finally selected for a more detailed analysis were classified into four groups. These groups, with representatives of each one of them, are introduced in the following.

2.1. Ad-Hoc solutions

This group comprises data mining applications that are specifically crafted to solve a concrete problem in a specific domain. The problem to solve is well-known from the beginning, and algorithms are selected and optimized around this precise problem. The developed data mining processes are then integrated into a software application with a user-friendly interface that hides the complexity of data mining techniques.

These solutions have been mostly developed for the educational [14,15] and medical/biomedical [16–19] domains. For instance, authors of [16] present a tool for the study of time-series data from haemodialysis treatments. The system allows clinicians to envision the evolution over time of different indicators, which are presented in 2D or 3D visualizations.

The problem these applications present is that their development and maintenance cost is quite high. This approach would be similar to hiring a data scientist to create a set of scripts that business makers can easily execute several times; but, when the script needs to be updated, the data scientist is again required. Moreover, tools developed this way tend to not be easily transferable to other problems or domains.

[1] The protocol that guided this review is available as supplementary material.

2.2. Workflow-based applications

Currently, there are several data mining tools that allow to quickly specify a data mining process by the definition of a workflow through an intuitive graphical interface. A workflow is a collection of interconnected building blocks, where each block represents a step of the data mining process, such as filtering data or executing an algorithm. Rapidminer [20], KNIME [21] and Weka [22] are well-known examples of this kind of tools.

Although workflows reduce the required time to define a data mining process, they are not oriented to non-expert users. Each building block has to be selected and configured for its precise task. Next, the blocks have to be appropriately interconnected to achieve the desired results. The parametrization of each block, as well as the orchestration of the workflow, are not yet simple enough tasks to be carried out by non-expert users.

2.3. Methodologies to develop non-expert-oriented applications

This category contains methodologies for the development of data mining applications that can be employed by non-expert users [23–26].

For instance, Zorrilla and García [23] propose to follow a service-oriented approach to democratise data mining. According to this approach, data mining processes are encapsulated into black-box modules that are offered as services. To prevent the user from having to configure these services, techniques like parameter-less algorithms [27] are used whenever possible. In other cases, these modules are configured to fit in with the specific needs of the domain. Then, web-based interfaces that access these services are developed. As details of the data mining techniques are hidden in the services, non-expert people should be able to use these applications.

Whereas *ad-hoc applications* are usually tailored to a specific data set of a concrete setting, these services aim to work with data sets coming from different settings of the same domain. For instance, following this approach, Zorrilla and García developed an application, called *eLearning Web Miner* [28], which is able to analyse courses hosted in e-learning platforms such as Moodle [29] or Blackboard [30].

These methodologies still make necessary the presence of experts, at least when they are initially deployed in a new domain. Nevertheless, they tend to benefit from the reuse of several components. For instance, the *core* of the services architecture presented in [23] could be reused in different domains. An additional problem that solutions of this category manifest is their flexibility to adapt to new analysis problems. For example, an application developed to analyse students activity could need severe changes to be able to analyse other entities of the same domain, such as student assignments or exams.

2.4. Generic applications for non-expert users

These applications [31–33] offer high level solutions to execute some tasks of a data mining process without the intervention of an expert. A representative of this group is *Oracle Predictive Analytics* [31]. This solution provides several data mining procedures that can be applied to data contained in a table from an Oracle database. An example is the `PROFILE` procedure, which tries to find the reasons behind a target column from a table taking a certain value. So, this command could be used to find the main reasons behind students getting `fail` as a value for a `CourseOutcome` column, i.e. the reasons because students fail. With this information, teachers may adopt corrective actions to improve course performance.

These procedures are black-box elements, and they do not require any special configuration. Therefore, they can be applied to different domains indistinctly. Nevertheless, since they are not tuned for the particularities of each domain, their accuracy can be affected considerably. Moreover, this approach does not cover all the stages of a data mining process, as the user is still responsible of obtaining the data, arranging it as a table, cleaning it and visualising the results of the procedures properly.

Similarly, [32] offers a wizard, which is integrated with the RapidMiner workflow tool, to automatize the task of selecting an algorithm in a data mining classification process. A ranking of the most promising classifiers is shown to the user, who selects the algorithm to apply. Nevertheless, the organization and configuration of the remaining blocks of the workflow is left to the application user.

2.5. Review conclusions

Table 1 offers a summary of the groups described in the previous paragraphs, including the main characteristics discussed for each group: development cost, accuracy of the applied data mining processes, and level of expertise in data mining required to make use of the solutions.

As we can see, cost remains low both for workflows and generic applications. One reason for this is that these groups are domain-independent and, as such, they do not need to be adapted to any domain specificities. However, this independence provokes the following issues: for workflows, although they allow fine-tuning data mining processes to achieve high accuracy, they remain as tools only usable by experts in data mining; as for generic solutions, they are domain-agnostic black boxes usable by non-experts, but their generality induces that the obtained results might not be accurate for some contexts.

Table 1
Summary of the research approaches in the data mining for non-experts field.

Group	Cost	Accuracy	Expertise required
Workflows	Low	High	Yes
Ad-hoc solutions	High	High	No
Development methodologies	Medium-high	Medium-high	No
Generic Applications	Low	Low-medium	No

```
Q1 show_profile of Students;
Q2 show_profile of Students with courseOutcome=fail;
Q3 find_reasons_for courseOutcome=fail of Students;
```

Fig. 1. Examples of queries written with the educational DMDL.

Finally, the development methodologies present solutions that benefit from a certain level of reuse between concrete developments, which may contribute to slightly reduce development costs. Also, these methodologies allow adapting the analysis processes to the applied domains, achieving reasonable accuracy results while maintaining the application friendly enough to be used by non-experts.

After this review of the state-of-the-art, we combined ideas coming from the second and third categories, i.e. *Methodologies to Develop Non-Expert-Oriented Applications* and *Generic Applications for Non-Expert Users* respectively, to overcome some of their limitations. First of all, we decided we would create a solution with high-level analysis primitives that would abstract from technical details. The primitives should be usable by business experts without expertise on data mining techniques. Moreover, these primitives should work with different entities of the same domain, achieving generality. However, we did not want to sacrifice accuracy in favour of generality, so these primitives should be adaptable to the particularities of each domain.

The solution we initially devised was the following: we would apply techniques of Domain-Specific Languages (DSLs) [34–37] in order to create high-level query languages (DMDLs, as introduced in Section 1). These languages, tailored for specific domains, would allow business experts to specify data mining tasks over data of their domain. The languages composition would consist of high-level commands, such as the procedures contained in the *Oracle Predictive Analytics* case [31], to execute data mining processes, plus concepts and terms coming from the specific domain. Therefore, the language syntax should be familiar to experts in that domain. Queries written with these languages would be transformed into low-level code in order to execute the corresponding tasks. This execution would rely on black-box data mining components, such as the services provided by Zorrilla and Garcia [23]. As each language is tailored to a specific domain, the transformation process that executes the queries should be also adaptable for that domain, or even for each setting of it. This execution granularity would allow the incorporation of specific optimizations, improving accuracy.

To explore if our idea to develop DSLs for data mining was feasible, we created a DMDL for the educational domain. Next section describes this case study, and enumerates the reasons that motivated us to develop an infrastructure for the definition of similar DMDLs.

3. Motivation: experience from a DSL for the educational domain

The first DMDL we developed was devised for teachers to get insights of their courses by means of the analysis of the information stored in the e-learning platform, such as Moodle [29], that hosted each course. These insights could then be used to improve future course editions.

Two data mining processes were supported by this DMDL. The first one extracted profiles from a data set allowing, for instance, to separate students into different groups according to their characteristics. This way, a teacher might analyse what groups of students typically fail an assignment, and what are the most prominent features of each group. The second one tried to highlight the causes for a concrete event, such as a student dropping the course.

Fig. 1 shows some examples of queries written in this DMDL. The first query (Fig. 1, Q1) invokes the profile extraction process by using the primitive `show_profile`, with the objective of grouping the course students according to the information contained in a `Students` entity. This query might be used to know what kind of students are enrolled in a course, so that it can be adapted to their particular characteristics.

The second query profiles the students again, but just a subset of them (Fig. 1, Q2). The `with` keyword is used to define a filter that limits the analysis to those students who have failed the course (`course_outcome = fail`) in this case. The objective of this query is to find behaviours that led to this negative result.

The third query aims to find causes that explain why a student fails a course (Fig. 1, Q3). For this, the `find_reasons_for` primitive is used, where (`course_outcome = fail`) is specified as the event to be explained.

To execute the queries that can be specified using this language, several prebuilt data mining processes were created. These processes were coded as Java functions that relied on algorithms from the Weka [22] data mining library, and they were adequately optimised for the educational domain. Moreover, data sets for each one of the domain entities that can

be analysed using the DMDL were created. In this case, data were extracted from different sources, such as databases or activity logs of the e-learning platform. Then, these datasets were formatted in *ARFF*, the tabular format used by Weka.

This DMDL was developed following a model-driven approach [34,37,38]. First, its abstract syntax was specified with an *Ecore* metamodel [39]. Then, a concrete textual syntax was defined using *Xtext* [40]. The specified queries were transformed into executable data mining processes by means of code generation templates [41]. The code generation was performed with the *Epsilon* [42] model management suite. The generated code basically configured and invoked the data mining processes that were previously created.

Moreover, using the Xtext facilities, some additional features were added to the language. First, to ensure that the introduced query was correct, a query validator was developed. For instance, the domain entity name introduced in the query has to be valid, this is, a mapping between the provided entity name and an available dataset has to exist. Secondly, an autocomplete feature was created. This feature proposes existing primitives and domain terms to assist the user while writing a query.

The development of this DMDL revealed an important problem of our approach: its development might involve a considerable cost. However, we noticed that different DMDLs could share some commonalities. Therefore, the development cost of a new DMDL might be reduced by reusing components of previous ones. With this idea in mind, we analysed how reusable the components of the DMDL for the educational domain were. We noticed some of these components had problems to be reused because of the following reasons:

1. They were dependent on domain-specific elements.
2. They were highly coupled to other components, so they were affected by their changes.
3. Some of the components were responsible of too many concerns, which made them hard to modify.

To solve these shortcomings, the original components of the DMDL for the educational domain were refactored to create a more generic and modular infrastructure for the development of DMDLs, denoted FLANDM. This infrastructure provides a set of now generic components that can be easily configured to fit within a specific domain, reducing development effort, and, therefore, development costs. Next section presents this infrastructure.

4. FLANDM: a framework to develop DSLs for data mining

Here, we describe FLANDM, the solution we have developed to reduce development costs of DMDLs. This section is structured as follows: firstly, the case study that is used to illustrate our solution is introduced. Then, the definition of a DMDL is divided into two blocks: *queries specification* and *queries execution*. Queries specification block covers all aspects related to the writing of queries conforming to a well-defined syntax and employing a user-friendly editor. Queries execution block addresses the automatic transformation of these queries into low-level data mining processes that provide the desired results.

The question of determining whether it is worthy to develop a new DMDL for a given domain is not trivial. For this purpose, the set of guidelines proposed by Mernik et al. [43] might be helpful. As it has been commented, the main point in favour of DMDLs is their reduction of complexity through a syntax adapted for non-experts in data mining. However, the commitment to support a DMDL has to be considered in conjunction with its final users, i.e. the business experts.

4.1. Case study: indicator analysis of diabetes disease

The objective of this new DMDL was to analyse information from a group of patients that were tested for diabetes [44]. For each patient, along with the result of the test (positive or negative), data regarding different indicators, such as blood pressure, age, or diabetes occurrence in ancestors were collected. These data could be used, among other reasons, to find causes for the diabetes disease. The task that performs this analysis can be invoked with the following query: `find_reasons_for test_result = positive of Diabetes_Results`.

Other possible analyses could involve the classification of patients in groups according to some features, or the ranking of most relevant indicators when determining if a patient has or does not have diabetes.

Next sections describe how this DMDL can be developed using FLANDM. It is shown how FLANDM allows the systematic and controlled reuse of several components initially developed for the DMDL for the educational domain. Those elements from the original DMDL that were modified to support this reuse are pointed out.

4.2. Queries specification

This section focuses on the development of a syntax and an editor for the specification of queries. We are again following a metamodeling-based approach [34], so the first step involves the definition of an abstract syntax for the DMDL to be created. As previously commented, the syntax of our DMDLs has elements of two different natures: (1) commands that determine the analysis tasks to be performed, and (2) terminology that refers to domain entities, or attributes of these entities, that is used as input for the analysis task. In the diabetes case study presented in Section 4.1, the entities would correspond to the patients diabetes data, and their attributes would be the gathered indicators and the final result of the test.

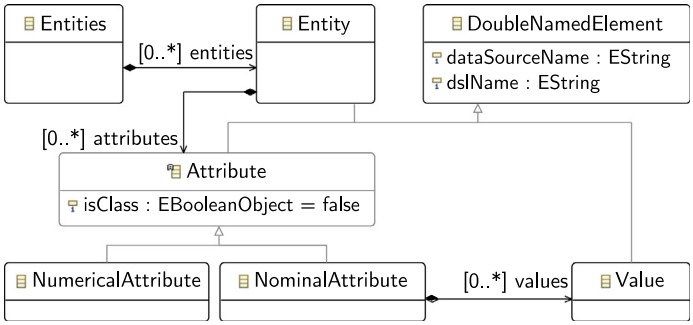

Fig. 2. Entities Metamodel.

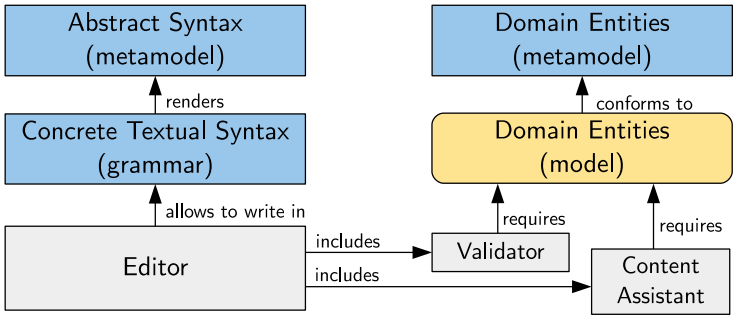

Fig. 3. Overview of FLANDM syntax and editor components.

The first set of elements, i.e. the commands, are domain independent. However, the second set, regarding domain terms, depends on each domain and, consequently, needs to be modified each time a new DMDL is developed.

When trying to adapt the structure of the DMDL from the educational domain for its reuse in the diabetes case study, we noticed that the management of the second set of elements, this is, the domain terminology, was scattered across several components of the original DMDL. More specifically, they were present in the validation module that made sure the query introduced by the user was correct, and in the autocompletion system that assisted the user during query formulation. We also realised that the core logic of these components would not require modifications when used in a new domain. The only requirement was to have some sort of access to the accepted terminology from the domain.

To overcome this problem, the following solution was adopted in FLANDM. The presence of domain information was limited to a single component, the *entities model*. This model conforms to the corresponding *entities metamodel*, which is depicted in Fig. 2. As it can be seen, a `Domain` contains a collection of `Entity` objects. Each `Entity` has a name and a list of attributes. These attributes can be of different types, such as numeric (`NumericalAttribute`) or categorical (`NominalAttribute`). For the categorical attributes, also, the set of discrete `values` they might take is also registered.

Those elements in the entities metamodel that are named extend from the `DoubleNamedElement` metaclass. A `DoubleNamedElement` has two linked names: a user-friendly name for being shown in the language editor (the `dslName`), and the name that is used internally and in the data sources (the `dataSourceName`). This double-named nature allows an easy modification of concrete terms from the point of view of the editor, without affecting the internal usage of those terms. For instance, if an entity of the original diabetes data sources contains an attribute denoted as *bmi*, it could be given a more descriptive name for its usage in the editor (e.g. *bodyMassIndex*) through the `dslName` attribute, while avoiding the modification of the original sources (*bmi* would be used internally through the `dataSourceName` attribute).

After creating this metamodel, the original components of the DSL for the educational domain were refactored to take advantage of it. As a result, the structure of Fig. 3 was obtained. It represents the components of FLANDM's query specification editor. Each one of these components are described in the following.

4.2.1. Domain entities

According to the structure presented in the previous section, the first step to develop a DMDL for a new domain involves the creation of an entities model that specifies which entities are present in that domain along with the attributes these entities have. Unlike in the original DMDL, this step does not impact other components of the language, as the entities information is concentrated in a single model.

To identify and model these domain entities, we would need to perform a *domain analysis*. Several methodologies currently exist for this task [45,46]. FLANDM does not impose the use of any of these methodologies, therefore, DSL engi-

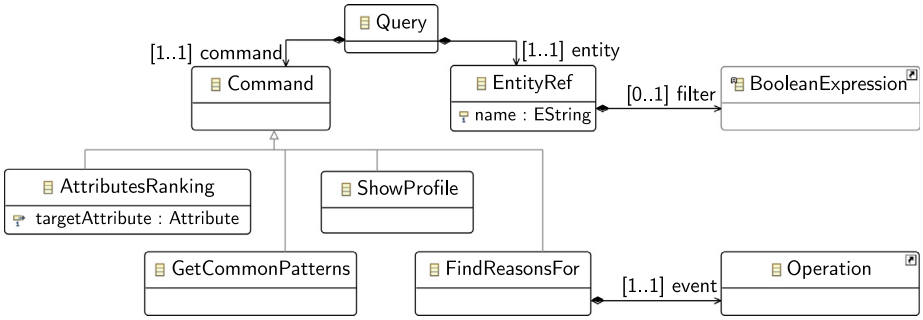

Fig. 4. Excerpt of the abstract syntax provided by FLANDM.

Table 2
Description of the analysis commands offered by FLANDM.

Command	Description	Attributes
ShowProfile	Organises the entity instances on different groups with similar characteristics.	–
GetCommonPatterns	Looks for frequent patterns present in the entity instances.	–
AttributesRanking	Returns an ordered list of the most strongly related attributes to the one selected as target.	Target: attribute to rank
FindReasonsFor	Finds possible causes of the presence of the indicated event in the data.	Event: phenomena to investigate

neers might use their preferred one. In general, we have opted for using well-known techniques of Domain-Driven Design [47] when creating the entities model of a new DMDL, since we have previous experience using them.

The defined entities model also determines the datasets that must be made available for posterior analyses. Precisely, each entity in the model has to be linked to a concrete dataset. The process of extracting and formatting domain data into these datasets is outside of the scope of this paper, as FLANDM does not offer any support for it. This process is usually carried out by data engineers, who perform the data extraction and cleaning through a set of low-level scripts [3]. However, we are working in another language, Lavoisier [48], for providing non-experts with data selection mechanisms over high-level models of the domain information.

4.2.2. Abstract syntax

Next, the abstract syntax for the DMDL must be defined. To support this step, FLANDM provides a base abstract syntax that can be customised according to the needs of each user. Fig. 4 shows an excerpt of this base abstract syntax. The `Query` metaclass is the root element of this syntax. Each `Query` has a `Command`, which indicates the analysis task to be performed.

FLANDM provides an initial set of commands. Table 2 shows a description of some of these commands. As it can be observed, some commands require extra arguments to work. For instance, the *FindReasonsFor* command requires the *event* to be explained as an argument.

Therefore, when developing a new DMDL, we only need to select those commands that correspond to the analysis task that experts of the target domain want to execute. It is worth to remember that these commands are domain independent, so they can be easily reused across domains. If a new analysis task had to be incorporated to fit in with the needs of a new domain, we would only need to create a new command that extends from the `Command` metaclass.

Each query is executed over a domain entity. This entity is captured in the abstract syntax through an `EntityRef`. An `EntityRef` contains a name that points to an existing domain entity. Moreover, it is possible to focus the analysis on a specific subset of the entity instances by providing a `filter`. A filter specifies that only those instances of an entity that satisfy a boolean expression must be used as input for a command.

In summary, to define an abstract syntax for a new DMDL, just the following actions are required:

1. Select those commands that will be used for the analysis tasks.
2. Provide new commands that extend the `Command` metaclass, if new analysis tasks are required.

4.2.3. Query validator

We have seen that by using entity references, we decouple the syntax specification from the domain entities. However, any text might be provided as an entity name, because it is a free string field. The same happens in the case of other free fields of the syntax, such as attribute names or values. Therefore, it can be the case that the user provides a name that does not correspond to any existing domain term, or that they simply misspell the name. To check that only valid domain terms have been provided, we developed an external syntax validator. This validator makes some complementary checks, in addition to the ones that ensure a query is syntactically correct. For instance, it checks whether the name of an entity reference actually corresponds to an existing domain entity. In addition, more fine-grained controls are also carried out, such as that attributes specified in a filter really belong to the domain entity being analysed; or, if an operation defined

```
00  @Check
01  def checkEntityName(EntityRef entity) {
02    if (!entitiesProvider.entities.exists[
03          e | e.dslName.equals(entity.name)
04        ]) {
05      error("Entity with name '" + entity.name +
06          "' does not exist",
07          QueryLanguagePackage::eINSTANCE.entityRef_Name)
08    }
09  }
```

Fig. 5. Check function which validates the name of an entity.

$$\langle query \rangle \quad ::= \langle command \rangle \text{ `of'} \langle entityRef \rangle \tag{1}$$

$$\langle command \rangle ::= \langle showProfile \rangle \tag{2}$$

$$\mid \langle findReasonsFor \rangle \tag{3}$$

$$\mid \langle attributesRanking \rangle \tag{4}$$

$$\mid \langle getCommonPatterns \rangle \tag{5}$$

$$\langle entityRef \rangle ::= \langle name \rangle \ (\text{`with'} \langle booleanExpression \rangle)? \tag{6}$$

$$\langle showProfile \rangle ::= \text{`show\_profile'} \tag{7}$$

$$\langle findReasonsFor \rangle ::= \text{`find\_reasons\_for'} \langle operation \rangle \tag{8}$$

$$\langle attributesRanking \rangle ::= \text{`attributes\_ranking\_for'} \langle attribute \rangle \tag{9}$$

$$\langle getCommonPatterns \rangle ::= \text{`get\_common\_patterns'} \tag{10}$$

Fig. 6. Base grammar offered by FLANDM for the DMDLs.

over an attribute makes sense regarding its type. As an example, the validator would ensure that a *greaterThan* operator is applied only to numerical attributes, indicating an error when it is employed with categorical ones.

This validator was implemented by using the facilities of *Xtext* [40], which is the software employed in the concrete textual syntax. Xtext offers the possibility to define check functions in Java or in the Xtend language, which are triggered over different elements from the grammar. As an example, Fig. 5 shows a validation function that checks whether the provided entity reference actually corresponds to a domain entity defined in the entities model. To do this, the function first looks for the provided name among the existent entities in the model. This model is accessed through the `entitiesProvider` object (Fig. 5, lines 02–04). If the name is not found, the validator raises an error with a proper explanation message (Fig. 5, lines 05–07).

This validator has been implemented following a domain-independent approach. It uses the domain entities model, but changes in this model does not affect to the validator, as it accesses the model through the `entitiesProvider` element, which is also generic.

4.2.4. Concrete syntax

After defining the abstract syntax, a conforming concrete syntax must be provided [34]. In FLANDM, a concrete textual syntax is defined through an Xtext grammar [40]. As with the abstract syntax, a base grammar provided as a base for new DMDLs. Fig. 6 shows an excerpt of this base concrete syntax in EBNF (*Extended Backus-Naur Form*) format.

According to this syntax, a query starts by introducing a command keyword, which determines the selected `Command` for the analysis. For instance, the `find_reasons_for` keyword indicates the use of the `FindReasonsFor` command. Any extra required information for each command is specified next to the keyword, such as the `operation` that represents the analysed `event` of the `FindReasonsFor` command. Then, the information about the entity to be analysed is provided. First, its `name`, which is a simple string literal, is introduced. Optionally, it is possible to define a filter by using the `with` keyword plus a `booleanExpression`, as commented before (see Fig. 1).

Xtext allows the definition of new languages by extending a base one. This feature is exploited in FLANDM, as new DMDLs extend the grammar of Fig. 6 for the definition of their own concrete syntax when needed. It should be noticed that the grammar of Fig. 6 is, thanks to the use of the entities model, domain-independent again. Therefore, starting from this grammar, we can define the concrete syntax for a new DMDL following an incremental approach [49]. The steps would be as follows:

```
00 override public void completeAttribute_Name(EObject model,
01       Assignment assignment, ContentAssistContext context,
02       ICompletionProposalAcceptor acceptor) {
03     val entity = entitiesProvider.findEntityByModel(model)
04     if (entity == null) { return }
05     for (attribute : entity.attributes) {
06       acceptor.accept(createCompletionProposal(
07           attribute.dslName, context))
08     }
09 }
```

Fig. 7. Assistant function that suggests attributes of an entity.

- Provide new production rules, like the ones depicted in Fig. 6, Lines 7–10, for the new commands that might have been added to the abstract syntax and grammar, if any.
- Select those commands that are to be finally included in the new DMDL, by modifying the grammar rule of lines 2–5.

4.2.5. Auto-completion

Finally, to make the generated editor more user-friendly, an auto-completion assistant was created. This assistant provides term proposals to the user when writing a query. It was again developed using the Xtext facilities.

As in the case of the validator, we refactored this module from its original form, so that it accesses the domain entities model, but changes in this do not affect its logic, which makes it domain independent. Therefore, this feature can be reused without changes in different domains, and so it was for the DMDL for the diabetes data.

As an example, Fig. 7 shows the function that suggests attribute names when the user wants to define a filter over an entity. For that purpose, this function first looks for the entity in the model (Fig. 7, line 03), and then transverses its attributes list generating a proposal for each one of them (Fig, 7, lines 05–08).

4.3. Queries execution

The execution of DMDL queries is the step where these queries get a semantic meaning. From the different ways of formalizing semantics described by Kleppe [34], we make use of the *translational* approach, in which programs of the developed DSL are translated to a language with well-defined semantics, such as C or Java. In our case, DMDL queries get a meaning by employing them to generate analysis scripts that make use of a data mining library.

The translation process provided by FLANDM is the result of refactoring the execution components of the original educational DMDL. As a running example to illustrate the elements described in this section, we use a query from the DMDL for diabetes data: `find_reasons_for test_result = positive of Diabetes_Results`. This query looks for causes of obtaining a positive result in the test according to the information captured from the patients.

Each query of the educational DMDL (see Section 3) was computed by transforming it into a data mining process and executing that process. Specifically, queries were converted into Java code through code generation templates. These templates worked in a *one-step-does-all* monolithic fashion, without a clear separation between transformation stages. This monolithic structure, although simple and straightforward, hinders reusability. Precisely, this structure implies that code generation templates are dependent on the following components: (1) the language syntax, which may vary for different DMDLs; (2) domain specificities, as the template has to know how to obtain the data associated to each domain entity; and (3) the employed data mining libraries, because the code generation targets a concrete analysis solution, e.g. Weka.

It should be also noticed that the low-level data mining processes that are finally executed for computing each command might vary depending on the application domain. For instance, the `ShowProfile` command might be translated into the execution of the Xmeans [50] clustering algorithm, whereas for other domains, other algorithms, such as DBSCAN [51], might provide more accurate results. Moreover, it might be the case that a switch in the underlying data mining platform is desired. For instance, for some algorithms, we might prefer to rely on RapidMiner instead of Weka.

The original monolithic structure caused that the code generation templates were affected by all the changes described above. To avoid this problem, when developing FLANDM, we separated the transformation process into two stages, as depicted in Fig. 8. Instead of directly generating Java code from the introduced query, we first transform each query into an abstract data mining task. We developed a *Data Procedure* metamodel for the specification of this kind of tasks. This metamodel allows the definition of platform-independent data-mining processes. Then, these processes are transformed into code that invokes algorithms of one or more data analysis platforms, such as Weka or RapidMiner. As in the case of the educational DMDL, this transformation process is completely transparent to the final user, who only introduces the query and receives the results of computing it.

The data procedure metamodel and the two new transformation steps are described in the following.

4.3.1. Data procedure metamodel

The metamodel diagram is shown in Fig. 9. A `Procedure` defines an abstract data mining task. All procedures operate over a `DataSource`, which is the equivalent to the `Entity` metaclass of the languages' abstract syntax metamodel (Fig. 4). This procedure metaclass is the root of a hierarchy that represents data mining methods. Intermediate metaclasses group

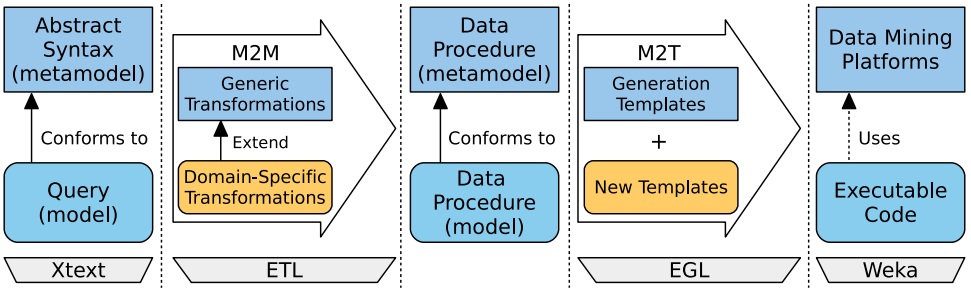

Fig. 8. Two-step query transformation process: a model-to-model (M2M) transformation is followed by a model-to-text (M2T) code generation phase.

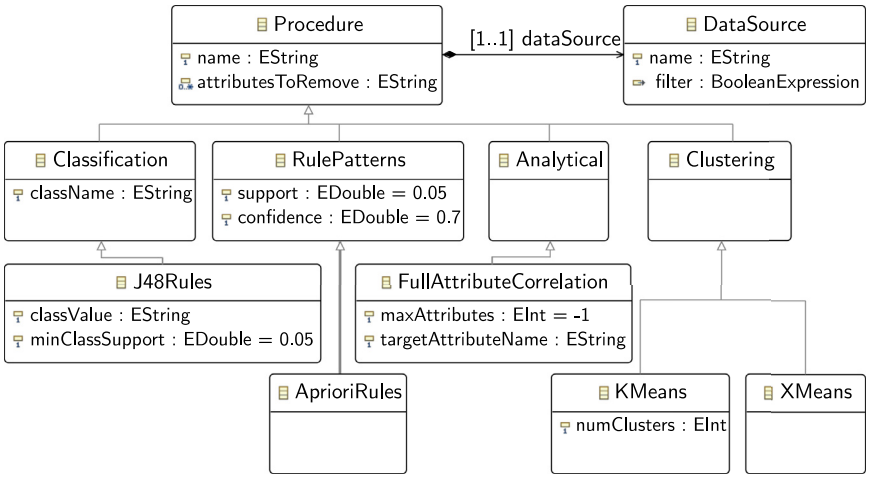

Fig. 9. Data procedure metamodel.

data mining techniques of the same family, such as classification or clustering, and they contain elements that are shared by all members of the family. For instance, the `Classification` metaclass, which represents the family of all classification methods, has an attribute named `className`, that identifies the attribute of an entity that is used to label its instances. The leaves of this hierarchy represent concrete data mining algorithms. Each leaf can have extra attributes that correspond to parameters for that algorithm.

For instance, let us consider the `J48Rules` procedure, which is a classification one. It requires the provision of two extra attributes: `classValue` and `minClassSupport`. This procedure constructs a J48 decision tree [52] over the entity's class attribute specified by the `className` field, which is inherited from the `Classification` metaclass. Then, the procedure traverses the branches of this tree and selects those ones that lead to a concrete value of the class, determined by the `classValue` attribute. Finally, the selected branches are formatted as if-then rules and shown to the user. It is possible to filter out those rules that do not represent a solid enough portion of the instances set. This filtering can be accomplished through the `minClassSupport` attribute, which has a default value of 0.05. This value indicates that a rule has to be applicable to at least 5% of the entity instances in order to be shown in the results.

4.3.2. Query to data procedure transformation

The first step in the transformation process is the translation from a query specification to a platform-independent data procedure. The input of this translation is the model of the introduced query (Fig. 8, left). This model conforms to the abstract syntax metamodel described in the previous section, and it is generated by Xtext after parsing the instruction text written by the user in the query editor.

This translation is achieved by means of a *model-to-model (M2M)* transformation. The transformation rules are specified with the *Epsilon Transformation Language (ETL)* [53]. FLANDM offers a set of generic base rules that transform each one of the provided commands into a default data procedure. These default procedures can be modified by extending and/or overriding the base rules, either to tune some parameters of the default procedures, or to directly select other procedures that might be better adapted to the particularities of a specific domain.

Fig. 10 shows one of these base rules. In this case, according to its guard (Fig. 10, line 04), the rule transforms queries employing the `FindReasonsFor` command into invocations to the `J48Rules` procedure (lines 01–02). This rule extends a more abstract `Query2Procedure` rule (line 03), which sets the attributes of the `Procedure` parent class, specifically, its `DataSource` attribute. Finally, the rule assigns values to the `className` and `classValue` parameters of the J48Rules

```
00 rule FindReasons
01  transform query : query!Query
02  to j48Rules : dataProcedure!J48Rules
03  extends Query2Procedure {

04  guard : query.command.isTypeOf(query!FindReasonsFor)

05  j48Rules.className =
      query.command.event.attribute.name;
06  j48Rules.classValue =
      query.command.event.comparison.value.name;
   }
```

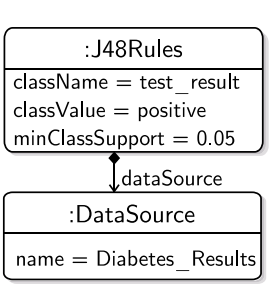

Fig. 10. Left: M2M transformation rule of a query in a J48Rules data procedure model; right: resulting J48Rules model of the M2M transformation over the example query.

```
   // data obtention
01 DataSource source = new DataSource(
       "data/diabetesResults.arff");
02 Instances ins = source.getDataSet();
03 ins.setClass(ins.attribute("test_result"));
   // rules generation
04 J48 j48 = new J48();
05 j48.buildClassifier(ins);
06 RuleSet ruleSet = j48.toRules();
07 ruleSet = ruleSet.filterByConsequent("positive");
08 ruleSet = ruleSet.filterByClassSupport(0.05);
   // results visualization
09 RulesVisualizer.show(ruleSet);
```

```
IF (body_mass_index > 29.9 AND
    plasma_glucose_concentration > 157)
THEN result = tested_positive

Support: 11.979%, Confidence: 86.957%
```

Fig. 11. Left: resulting code of the M2T generation applied over the example procedure model; right: an example of the rules obtained when running the generated code.

procedure (lines 05–06). These values are extracted from the `event` attribute of the `FindReasonsFor` command. For this case, the `minClassSupport` attribute is left unmodified, so its default value of 0.05 will be used. The resulting data procedure model is shown in the right of Fig. 10.

In summary, FLANDM offers a set of M2M transformation rules for the query to data procedure translation step. These rules are domain-agnostic, which may induce a less than adequate accuracy of the results for some specific contexts, such as what happens with the Oracle Predictive Analytics stored procedures [31]. Nevertheless, this is a known issue in the data mining field [54]. Therefore, this step of the queries execution was designed with extensibility and modifiability in mind, as the provided rules can be adapted, with the help of an expert, to any specificities of the target domain for achieving better results.

4.3.3. Data procedure to code transformation

The last step of the query transformation process involves a *model-to-text (M2T)* transformation phase, implemented by code generation templates specified with the *Epsilon Generation Language (EGL)* [55]. These code generation templates convert each data procedure specification into code for invoking a proper implementation of the algorithm represented by the procedure, which are typically provided by a data analysis platform.

As in the case of the M2M transformation, FLANDM offers a set of M2T generation templates. It should be noticed that these templates just transform a platform-independent specification of a data procedure into code. Since these platform-independent specifications are precisely detailed, no new decisions depending on the domain need to be adopted. Thus, these templates are domain-independent and can be reused with no changes in different domains.

The Java code obtained when performing the M2T transformation from the data procedure model depicted in Fig. 10, right can be seen in Fig. 11, left. The literal values appearing in the code are obtained from the data procedure model obtained through the M2M transformation. For the sake of simplicity, the concrete EGL templates are left out of the paper, as they contain some burdensome implementation details that are unnecessary and detrimental for the general explanation of the translation. In this piece of code, firstly, the selected dataset is loaded as a Weka's `Instances` object ((Fig. 11, lines 01–02). Then, the attribute `test_result` is selected as class attribute (from the procedure's `className`), and a `J48` tree is built over the instances data (lines 03–05). Classification rules are extracted from this tree and, from those, the ones that have a `positive` value in the consequent and a high-enough class support are selected (lines 06–08). For these selections, the attributes `classValue` and `minClassSupport` of the `J48Rules` procedure are employed. Finally, the resulting rules are shown to the user (line 09).

In Fig. 11, right we can see one of these rules. The rule indicates that if the `plasma_glucose_concentration` and `body_mass_index` indicators of the patient exceed some limit, there is a high probability that the result of the diabetes test is positive. For each rule, metrics about its support and confidence are provided. The support indicates that

these excessive indicator values happen in about 12% of the analysed data instances, of which, in ∼87% of the cases, the test result was positive. This information could be used by clinicians to preventively perform a diabetes test to those patients who show the mentioned indicator excesses in regular check-ups.

Concluded the definition of FLANDM's structure, next section evaluates the benefits of using FLANDM through several language definitions.

5. Evaluation

The main objective of FLANDM is to reduce the development costs of DSLs for data mining democratisation (DMDLs). This section evaluates how far this objective has been satisfied. Moreover, it also evaluates a second benefit, which states that the use of FLANDM also reduces the maintenance costs of these DMDLs. The two issues were measured separately. First, we analysed how FLANDM helps to reduce development costs by the definition of four DMDLs with this framework. Next, the contribution of FLANDM in the reduction of maintenance costs was evaluated by analysing how these DMDLs were affected by several scenarios, which required to perform some modifications on the languages. The four DMDLs developed as case studies, the evaluation process, and its results are described in the following sections.

5.1. Case studies

This section describes the four DMDLs that we created to evaluate our work. We relied on third-party case studies coming from heterogeneous domains. This should help to avoid bias and to increase the external validity of our analysis, as well as to demonstrate that FLANDM is applicable to different domains.

The four DMDLs that were developed during this evaluation process are:

1. A language for the educational domain, which analysed data stored in e-learning platforms (Section 3).
2. A language for the medical domain, which analysed results of diabetes tests (Section 4.1).
3. A language for the business management domain, which analysed business reviews coming from the *Yelp*[2] platform.
4. A language for the public administration domain, which analysed data about visa request processes of the *American Department of Labour*.

The first and second case studies have already been introduced in this paper (see Sections 3 and 4.1, respectively).

The third language was designed to analyse data gathered from the Yelp platform. Yelp provides a business review service for customers to write their opinions and for owners to describe and advertise their businesses. Yelp has proposed different data analysis challenges[3] in the last few years. In each challenge, some actual data is made publicly available, and a contest is organised to discover hidden insights in these data that might be of interest for Yelp. Available data include information about users, business reviews, businesses features (e.g. Wi-Fi, vegan food, parking), among others. The DMDL aims to provide a high-level language to answer some of the questions proposed in these contests.

The fourth language analyses data released by the American Department of Labour[4] about the processing of work visa requests it receives. There are different foreign workers programs, which vary in aspects such as the duration (permanent or temporary), the foreign worker's country of origin, or the degree of specialization of the work. For instance, the H-1B visa allows specialized foreign workers to obtain a temporary permit. Among other aspects, this visa requires an approved sponsoring employer prior to the filing of the petition. Each petition follows different status updates until it reaches a final outcome. The DMDL aims to find information about, for instance, what makes a visa request to be approved or denied.

To clarify how the third and fourth DSLs work, and since they have not been introduced in previous sections of this paper, Fig. 12 shows some examples of queries written with these DSLs.

The first query aims to find those business features that have a higher influence on the stars rating of the businesses from Edinburgh with a rating lower or equal to three stars. For this purpose, the `AttributesRanking` command is used. This command returns a list of attributes ordered by their correlation strength with the evaluated attribute (stars).

The second and third queries analyse the *H-1B_Visas* entity, which represents a specific type of work visa. The second query tries to find causes that made petitions in 2017 to be denied. The third query seeks for patterns in those petitions that either were approved (*status* = *certified*) or, despite its approval, were not finally completed (*status* = *certified-withdrawn*).

The order in which these DMDLs were developed is the same as the one used for the above description. This order has an influence on the development effort of each DSL, as those languages that were developed later could reuse elements that were created for prior DSLs. For instance, the first two DSLs made use of the initial `ShowProfile` and `FindReasonsFor` query commands. For the third DSL, two new commands, `AttributesRanking` and `GetCommonPatterns`, were created. The fourth DSL also made use of these commands, but as they were created before, it was possible to simply reuse them, which decreased its development costs.

[2] https://www.yelp.com/.
[3] https://www.yelp.com/dataset/challenge.
[4] https://www.foreignlaborcert.doleta.gov/performancedata.cfm.

```
Q1 attributes_ranking_for stars
     of Businesses
     with city equals Edinburgh
      and stars <= 3;
Q2 find_reasons_for status = denied
     of H-1B_Visas
     with year = 2017;
Q3 get_common_patterns
     of H-1B_Visas
     with status = certified
       or status = certified-withdrawn;
```

Fig. 12. Example queries of the business reviews (Q1) and visa approval (Q2, Q3) DMDLs.

Table 3
Parameters of the simple cost productivity model.

Name	Description
C	Obtained cost of developing a product through component reuse, relative to the development of the complete product from scratch.
R	Percentage of the original effort for developing a product that is avoided thanks to component reutilisation.
b	Cost of integrating a reused component in a product, relative to the development of that component from scratch.

5.2. Reduction of development costs

The main contribution of FLANDM is a reduction in development costs of a DMDL. This reduction is achieved thanks to the reutilisation of different modules, such as the autocompletion feature, the external syntax validator, or the code generation templates, among others.

This section measures the effectiveness of this reutilisation, regarding development effort, of the different modules provided by the FLANDM framework. To accomplish this objective, we relied on a third-party cost model for software applications based on reutilization. This model is described in the next section.

5.2.1. Reusability measurement method

As a measure of component reusability, we relied on the cost productivity model provided by Gaffney and Durek [56]. These authors provide a set of models to measure different issues of component reutilisation, such as forecasting when the effort of building a reusable component will pay-off. From this set of models, we used the *simple model*, which was the one that best fitted with our needs. The simple model aims to estimate what is the reduction cost obtained when some parts of a software application are built from reused components, which is how our DMDLs are developed. This simple model is based on several parameters, which are described in Table 3.

Parameter C aims to measure reusability effectiveness when developing a new product. A value of $C = 1$ implies that the obtained cost through reutilisation of components is the same as the cost of developing that product without reusing elements, this is, as a completely new product. Therefore, reutilisation does not provide any benefit from a cost point of view in this case. However, reutilisation might provide other benefits, such as lower presence of bugs. When $C \leq 1$, reutilisation is cost-effective, whereas $C > 1$ indicates the opposite. Therefore, we are interested in obtaining a C value as low as possible.

The parameter b measures the required effort to integrate a reused component in the new system. Software modules can rarely be reused as they are, as some customisations and integration code needs often to be written. When $b = 1$, the effort required to integrate the components would be equivalent to developing those components from scratch. A value of $b < 1$ indicates that the integration of a component was cheaper than developing it, whereas $b > 1$ means the opposite. Thus, we are interested again in getting values of $b \leq 1$ and as low as possible.

R specifies which portion of the development effort required for building a product from scratch can now be saved due to component reutilisation. For instance, if we are reusing a couple of components whose development effort, if these components were developed from scratch, is 20% of the whole product, then $R = 0.2$.

$$C = (1 - R) \cdot 1 + R \cdot b = (b - 1) \cdot R + 1 \tag{1}$$

With these definitions, to calculate the value of C, Gaffney and Durek define the Eq. (1). In this equation, "1" represents the cost of developing a product completely from scratch. So, $(1 - R) \cdot 1$ represents the cost of developing those parts of a software product that are new, whereas $R \cdot b$ adds the cost of integrating the components that are reused.

To clarify how this formula works, let us suppose we are developing an application where 50% of their development effort can be saved by reusing some prebuilt components. In addition, let the cost of integrating these components be a half of the cost of developing them from scratch. These premises imply that $R = 0.5$ and $b = 0.5$, so $C = 0.75$, which means we have saved a quarter of the whole development effort thanks to saving a half of a half of the original development effort.

Table 4
Stages of a DMDL development using FLANDM with their estimated weights.

Step	Name	Weight (%)
S_1	Domain definition	45
$S_{1.1}$	Data acquisition	35
$S_{1.2}$	Domain entities definition	10
S_2	DSL editor development	30
$S_{2.1}$	Abstract syntax definition	10
$S_{2.2}$	Complementary syntax validations	5
$S_{2.3}$	Concrete syntax definition	10
$S_{2.4}$	Auto-complete development	5
S_3	Query execution	25
$S_{3.1}$	Query translation to DM processes	10
$S_{3.2}$	Platform code generation	15

Table 5
Lines of code (LOC) parameters used for the definition of b.

Name	Description
MLOC	*Modified* lines of code when adapting a FLANDM component to a new domain.
CLOC	New lines of code written to *customise* a FLANDM component for the new domain.
TLOC	Effective *total* lines of code of a component.

For the sake of simplicity, development effort is measured in lines of code (LOC). The use of this metric is debatable, but it is often, as in this case, the best available one without compromising objectivity [57].

In the calculations of the cost model parameters, we consider only in the implementation level of the development process of a DMDL. For the purpose of simplicity, other stages, like requirements elicitation, architecture design or testing are not considered. To estimate the effort associated to these stages can be highly complex and it is beyond the scope of this work.

To calculate the value of R, we firstly analysed what stages of the development process of a DMDL can be skipped by reusing componentes provided by FLANDM. Then, we provided a rough estimation, based on our experience when developing DMDLs, of the percentage of the global development effort associated to each stage. Table 4 shows these development stages and their corresponding estimations.

As it can be observed, the first step, S_1, is domain-specific. Consequently, its artefacts can be hardly reused across different domains. For instance, the data acquisition code for retrieving data from an e-learning platform and formatting it according to the ARFF format is specific for the educational DMDL, and it is not expected that it can be reused for other DMDLs. The same argument applies to the specification of the domain entities. For the remaining stages, S_2 and S_3, FLANDM provides components that can be reused as they are, or adapted depending on the particularities of each domain.

So, we consider the code produced in the stage *S1* as new code that needs to be written to create a DMDL, whereas stages *S2* and *S3* represent code that comes from reutilization. This means that the development effort associated to stage S_1 would be the $(1-R)$ term of the Gaffney and Durek's cost model, and the sum of the development effort of the other stages would be R. Therefore, $R = 0.55$ in our case, according to the weights of Table 4.

We based the calculation of b on three parameters, which are described in Table 5. *MLOC (Modified Lines of Code)* represents the lines of code that need to be modified when reusing a FLANDM component. For instance, some descriptions provided of the content assistant are often modified to adapt them to the particularities of each domain and thus become more user-friendly. *CLOC (Customisation Lines of Code)* counts how many lines of code were added to a FLANDM component for its usage in a new domain. For instance, to update how queries are translated into a data mining process, we need to write an ETL rule that overrides the default one. This code is considered customisation code. Similarly, if a new code generation template is added to support a new data mining platform, the whole template is treated as customisation code.

Finally, *TLOC (Total Lines of Code)* counts the lines of a FLANDM component that are effectively used in DMDL. It should be taken into account that some FLANDM components might have more lines of code than the ones that are actually used. For instance, the query execution component might have code generation templates for several target platforms, but just one platform is typically used. Therefore, to avoid noise, we just count those lines of a component that are really executed in a DMDL.

$$b = \frac{MLOC + CLOC}{TLOC} \tag{2}$$

Considering these premises, b is defined as in Eq. (2). This formula states that the relative effort for integrating a component or a set of components can be calculated as the sum of LOC that have been modified or newly written to adapt these components ($MLOC + CLOC$), divided by the total number of LOC of those components that are actually executed in a domain ($TLOC$). This is, if a quarter of the code of a component is modified or customisation code, $b = 0.25$, which means that the effort of integrating this component has been a quarter of the effort of developing it.

Table 6
Values of the b parameter for each step, weighted averages of b for each DMDL ($b_{S_{avg}}$) and final relative cost (C). Last row shows the average values of the four DMDLs.

	$b_{S_{2.1}}$	$b_{S_{2.2}}$	$b_{S_{2.3}}$	$b_{S_{2.4}}$	$b_{S_{3.1}}$	$b_{S_{3.2}}$	$b_{S_{Avg}}$	C
DSL_1	0.000	0.046	0.070	0.071	0.150	0.011	0.054	0.480
DSL_2	0.000	0.046	0.070	0.071	0.150	0.011	0.062	0.484
DSL_3	0.083	0.046	0.148	0.107	0.218	0.183	0.143	0.529
DSL_4	0.000	0.046	0.065	0.107	0.161	0.010	0.070	0.489
Avg	0.021	0.046	0.088	0.089	0.170	0.054	0.082	0.495

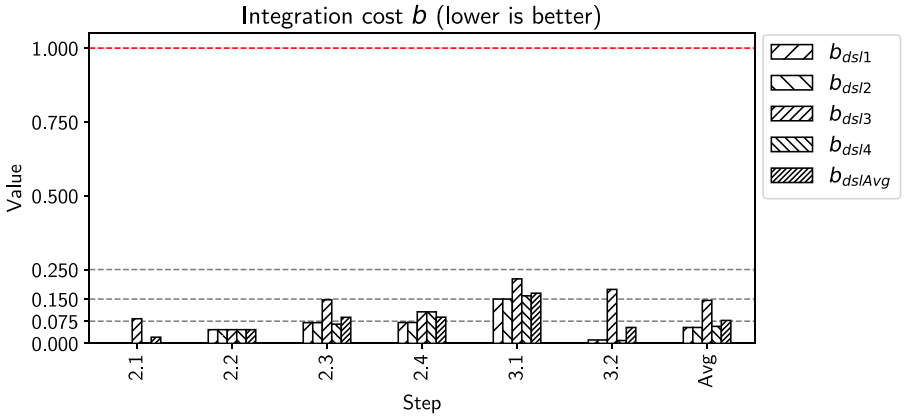

Fig. 13. Relative Integration cost (b) per development step of each implemented case study, plus its weighted average value (*Avg b*). The dashed line at value 1 specifies the critical point above which reutilisation is not cost-effective.

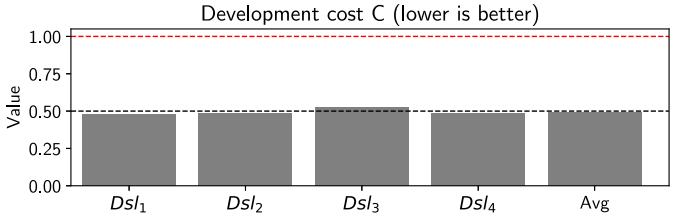

Fig. 14. Development cost C of the implemented case studies, along with the average value (*Avg*). Dashed line at value 1 marks the cost of creating each case study from scratch.

In our case, values of b were calculated for each development step where FLANDM components were reused. Then, a global b value was provided by calculating the weighted average of the b values obtained per step. As weights for the average, the estimated percentage of development effort provided in Table 4 were used.

5.2.2. Reusability results and discussion

Table 6 shows the gathered results after developing the four DMDLs previously commented. This table contains the values of b per development step of each DMDL. Besides, we provide the global value of b for each DMDL ($b_{S_{Avg}}$), computed as the weighted average of its indivual b values; and C, which indicates the relative cost of developing each DMDL, as compared to developing it completely from scratch. Figs. 13 and 14 depict these values graphically, so that they are easier to compare and visualise.

Fig. 13 shows the values of b, i.e. the relative integration cost, per each step of the development process. The dashed line in the top of this graph indicates the critical case where $b = 1$. In this case, reutilisation would not be effective from a cost point of view. Thus, we are interested in getting values of b placed as below as possible of this reference value. These values of b are shown for each developed DMDL (e.g. $b_{DSL_1} - b_{DSL_4}$). Besides, the average value of these bs per step ($b_{DSL_{Avg}}$) is depicted. Finally, the family of bars at the right (*Avg*) shows the global value of b for each DSL and the global average value of the experimentation.

These data show a considerable benefit from component reuse. The averaged global integration cost was 8.2%, which indicates that the effort for integrating reused FLANDM components was only an 8.2% percent of the effort that would have been required if developing these components from scratch. This is, approximately 90% of the development effort was saved.

These results show that FLANDM components are domain-independent enough to be integrated in new domains with very low effort.

As a consequence of this low integration effort, the averaged DMDL relative cost C was 49.5%. This implies that, due to the use of FLANDM, the development cost of a new DMDL is reduced, in average, by half. We discuss these results more in-depth in the following.

With respect to abstract syntax definition ($S_{2.1}$), the four DMDLs used the abstract syntax provided by FLANDM with no major modifications, which generated a low relative cost for its integration ($Avg(b_{2.1}) = 2\%$). The graph exhibits a pattern that repeats in other steps. The highest b value appears in the third language ($b_{DSL_3} = 8.3\%$). In this language, two new commands, i.e. `GetCommonPatterns` and `AttributesRanking`, were introduced. In the case of the abstract syntax, the `Command` metaclass of the abstract syntax had to be extended twice to create these commands. The fourth DSL also used these commands, but as they were already present in the abstract syntax, it could simply reuse them. Due to this reuse, the integration cost of the fourth DSL was low again. This phenomena reveals that b will decrease as the FLANDM framework grows and more and more elements can be reused. Therefore, cost effectiveness of FLANDM usage is expected to improve over time.

The integration effort of the syntax validator ($S_{2.2}$) exhibited the advantages of isolating domain elements in the entities metamodel. In the four cases, the base complementary syntax validator could be reused practically as is, which made its integration cost almost inexistent. About 95% of the original development effort was saved ($Avg(b_{2.2}) = 4.6\%$).

The integration cost of the concrete syntax ($S_{2.3}$) was similar to the cost of the abstract syntax. The base grammar provided by FLANDM was adapted to each case study with little difficulties. Although the averaged cost was higher ($Avg(b_{2.3}) = 8.8\%$) than in the abstract syntax, it remained consistently low across all the DMDLs, with the exception of the third language, because of the newly added commands, as previously commented.

Once again, the isolation of domain-specific elements highly benefited the integration cost of a component, in this case, the content assistant module ($S_{2.4}$). As in the case of the syntax validator ($S_{2.4}$), practically no changes were required ($Avg(b_{2.4}) = 8.9\%$). The only changes that were performed between domains, apart from basic configurations to glue components, took place in the commands proposal provider. When suggesting existing commands to use in a query, the assistant also shows a brief description of what each command does. To improve usability, these descriptions were adapted to better fit in with each domain.

Finally, regarding the execution of queries (steps $S_{3.1}$ and $S_{3.2}$), the usage of a data procedure metamodel as intermediate representation allowed the reuse of both default transformation rules and code generation templates across domains. However, some extra work was required to adapt the analysis for each domain.

Specially, this can be noticed in the values for step $S_{3.1}$, i.e. the transformation of the query into an abstract data procedure. In this step, data mining processes are often adapted to improve accuracy according to the specificities of each domain. This adaptation provokes that this step had the highest integration costs ($Avg(b_{3.1}) = 17\%$) when compared with other steps. However, this value, from a general perspective, remains low, indicating that components for this step can be reused with an integration effort of 20% of the cost of developing them as completely new modules. The code generation templates (step $S_{3.2}$) were free of these adjustments, and contained only target platform details. As there was no need to change the target platform, these templates could be reused with less effort.

The phenomena related to the addition of new commands appeared in these steps again. In DMDLs 1 and 2, two data procedures and its corresponding templates offered by FLANDM were reused (`Xmeans` and `J48Rules` specifically). For DMDL 3, two new analyses were introduced (`FullAttributeCorrelation` and `AprioriRules`) to give response to the newly defined commands. This increased the integration cost ($b_{DSL_3.S_{3.1}} = 21.8\%$; $b_{DSL_3.S_{3.2}} = 18.3\%$), as the framework had to be customised to incorporate two new data procedures, plus their corresponding ETL rules and code generation templates. However, in the case of DMDL4, these new templates were reused, and the b values improved accordingly ($b_{DSL_4.S_{3.1}} = 16.1\%$; $b_{DSL_4.S_{3.2}} = 1\%$).

Fig. 14 shows the relative development cost for each DSL. As before, the critical point above which reutilisation would not be cost-effective, i.e. $C = 1$ is depicted as a dashed line. The obtained results were very stable, around the 50% level. In the third case, which incorporated more changes, the relative cost was higher. However, this cost decreased again for the fourth language, showing that the opportunities of reuse when developing new DSLs increase as the FLANDM ecosystem grows.

These results show very good values taking into account the proportion of the final product that was actually reused. We were providing a framework for reusing the domain-independent components of a DMDL, whereas for the domain-specific part we did not provide any specific support. This means that we were providing support to 55% of the development effort, and achieving a relative cost reduction of $\backsim$50 %, which implies that the integration costs were really low.

Nevertheless, as the reader might notice, the value of R might vary depending on domain size, which would affect the final C. To clarify this issue, let us suppose the following two extreme cases. First, if we needed to analyse a toy domain comprised of just two small entities and whose data are stored in two clearly identified tables of a relational database, the effort associated to domain data definition might be really low. On the other hand, if we needed to deal with a huge domain containing hundreds of entities, whose data are retrieved using complex web scrapping techniques, the effort of domain data definition could be so high that the benefits of reusing the domain-independent part might be not noticeable.

To illustrate this issue, Fig. 15 shows the relation between R and C based on the Gaffney and Durek cost formula [56], and with the value of b fixed at 8.2%, the average value of our case studies. The point of this function corresponding to our case,

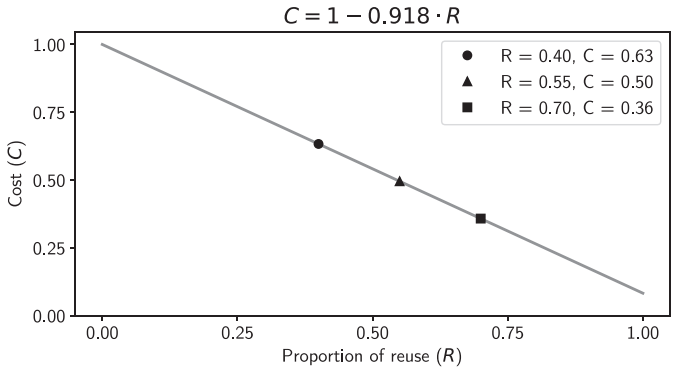

Fig. 15. Relation between DMDL cost C and proportion of reuse R, for $b = 8.2\%$.

where $R = 0.55$, is highlighted with a triangle. The points marked with a circle and a square represents two examples of cases the domain complexity increases and decreases. If the relative effort associated to the domain-specific part increased to a 60%, we might yet obtain a 27% of cost reduction, whereas if this domain-specific effort decreased to a 30%, we might save a 64% of the original development effort.

Therefore, it can be concluded that FLANDM reduces the cost of developing new DMDLs, by providing components that can be reused across different domains with low integration costs. These benefits might vary depending on the domain complexity but, even for relative large domains, it can still provide some noticeable benefits. Threats to the validity of these conclusions are analysed in next section.

5.2.3. Threats to validity

There exist some internal and external threats to the validity that might affect the previously discussed results. This section analyses how we have managed these threats.

5.2.3.1. Internal threats to validity. These threats are related with the method we used to measure reusability and some of the decisions we have adopted. First of all, we might have used an inaccurate cost model that biased our results. To avoid this, we have relied of a mature and well-known cost model, which have been widely used in the literature [58–64].

Secondly, it can be argued that the weights we have provided for the relative effort of each stage of the development process of a DMDL (Table 4) are rough and inaccurate estimations, that might present higher variations. These weights affect to the values of R and b.

In the case of R, as previously stated, this variation might be true due the relation of efforts assigned to the domain-specific and the domain-independent stages of the DMDL development process. Due to this fact, we have stated in the previous section that the value of C might vary depending on the domain complexity and size, which might make the benefits of our approach negligible for very large and complex domains.

Nevertheless, much lower variations are expected for the relative weights of each step of the domain-independent stages, i.e. *DSL Editor Development* and *Query Execution* (see Table 4), which influences the value of b. These weights vary proportionally in relation to the difficulty of each step. This is, the effort associated to the definition of the concrete syntax is expected to be approximately twice the effort for the development of the auto complete module. Since the relation between these weights remains stable, the weighted average of b values would not be affected. Moreover, since there are not high differences between the values of b for each stage, small variations in the weights would not produce noticeable changes in the results of the weighted average either.

In third place, the formula we have used to calculate b is self-elaborated, so it might be inaccurate or even wrong. To mitigate this threat, we paid special attention to the design of this formula, which was defined after a long process of refinement. Moreover, it was checked that the formula returned meaningful and reasonable results in different reuse scenarios.

Fourthly, for the sake of simplicity, we simplified the estimation of the development effort of a DMDL focusing just on the implementation stage. Other phases were simply ignored, so this might have an effect in our results. Regarding this issue, it should be noticed that the cost of some of these phases might also be reduced thanks to the use of FLANDM. For instance, a lower testing effort is expected since reused components do not need to be completely tested at the unit level; an analysis of their modifications and of the integration of the component with the whole system would be sufficient. For other stages, such as requirements elicitation, the associated cost is expected to be similar. In any case, the cost is not expected to be higher by the use of FLANDM because of these development stages. Therefore, these simplifications do not invalidate our conclusions.

We must also consider how appropriate is the use of Lines of Code (LOC) for measuring effort. This is a classical effort metric whose accuracy has been largely discussed. There are three main drawbacks associated to this metric:

Table 7
Change scenarios.

Id	Name
ChSc1	Syntax renaming or translation
ChSc2	Domain entities evolution
ChSc3	Customisation of the query transformation process
ChSc4	Addition of new commands

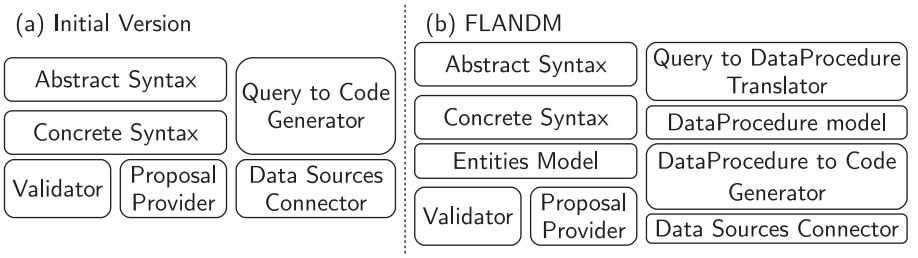

Fig. 16. DMDL architecture without (a) and with (b) FLANDM.

1. The effort for producing a line of code might vary between languages.
2. Two pieces of code of equal size and written in the same language might have a different cost because one is conceptually more complex than the other one.
3. Two pieces of code written in the same language and with the same functionality might vary in size due to different coding styles.

It should be noticed that LOCs are used to calculate the values of *b*. This parameter is calculated per each development stage. Therefore, when calculating *b*, we are mostly taking into account LOCs coming from the same language, e.g. ETL in step 3.1 of Table 4, and with the same conceptual complexity. Moreover, the differences between the conceptual complexity of each stage are considered in the weights associated to each stage (see Table 4). These facts contribute to mitigate issues 1 and 2. Regarding the third use, we have ensured, before measuring LOCs, that all pieces of code followed the same coding rules.

Finally, we acted as DSL developers when implementing the DMDLs for the evaluation. Obviously, we are experts in the structure of FLANDM, so it may be said that the cost reduction offered by FLANDM is due to this expertise. Nevertheless, the evaluation results are expressed in relative terms, which are applicable independently to the expertise of the developer of the DMDLs. For instance, a newbie DSL developer may take more total effort to develop a DMDL by employing FLANDM than we did. However, if FLANDM helps reduce this effort by ∽50% as stated, then the reduction for this developer would be that 50%, although of a larger total effort.

5.2.3.2. External threats to validity. Related to external threats, it might be that the results were influenced by some particularities of the four analysed DSLs. To mitigate this issue, we relied on third-party case studies, over which we do not have influence, and which came from heterogeneous domains.

5.3. Reduction of maintenance costs

With respect to maintenance costs, the usage of DSLs might be beneficial, as domain experts have more hands-on control in the created code; but also it could increase these costs if, as stated by Deursen and Klint [65], the DSL itself needs to be updated frequently. In our case, the whole FLANDM environment can be considered as an external DSL, whereas each DMDL derived from FLANDM might be viewed as internal DSL embedded in the FLANDM architecture [8,9]. The definition of internal DSLs brings some benefits, such as reuse of base language elements, but it might also lead to maintenance problems, since changes in the base language might have ripple effects on the internal DSLs defined on it.

Therefore, we took care of avoiding these ripple effects. To do it, we put a special effort during the refactoring process of the original DMDL structure that took place during the development of FLANDM, in order to design a loosely-coupled, modular DMDL architecture, which is used as base for the definition of new DMDLs. FLANDM's architecture should contribute to a better evolution and maintenance by limiting the impact of changes.

To illustrate and evaluate this complementary benefit, this section discusses a set of change scenarios that we have typically faced. These scenarios can be found in Table 7. The changes range from purely aesthetic fine-grained issues (e.g. changes in the syntax (*ChSc1*)) to the addition of new coarse-grained elements (e.g. support new commands in the DMDL(*ChSc4*)). For each scenario, we compared how the original architecture of the DMDL for the educational domain and the new architecture of FLANDM were able to deal with the required changes. Both architectures are summarised in Fig. 16. For simplicity, in the following, we refer to the architecture of the DMDL for the educational domain as the *initial version*.

5.3.1. ChSc1: syntax renaming or translation

Change scenario *ChSc1* was motivated by a request of some teachers about translating the syntax of the DMDL to Spanish, which was their mother tongue. This change scenario comprises two different kind of changes: (1) renaming the domain-independent part of the DMDL, i.e. command names and other keywords such as *and* or *with*, and (2) renaming its domain-specific part, which was comprised of names of domain entities and their attributes.

Regarding renaming of domain-independent parts, the changes can be easily accommodated in both versions of the DMDL by just changing the concrete syntax definition, thanks to the separation between abstract and concrete syntax. Moreover, to address completely this issue, error messages shown by the syntax checker and the auto complete feature were also translated. In summary, the effort was similar in both cases. This was expected, since this benefit comes from the metamodeling-based approach followed for the language definition in the two versions.

On the other hand, the second kind of changes involved different tasks between the versions. In FLANDM, the changes were limited to the entities model component, due to the double-named nature of elements (see Section 4.2). In the original version, this change scenario was not foreseen, and the domain elements were scattered across different components. More specifically, a change in a term affected the *validator*, the *autocompletion*, and the *data source connector* components.

5.3.2. ChSc2: domain entities evolution

Domains evolve, which means that new domain elements might need to be added or existing ones removed.

Adding a new entity, or a new attribute to an existing entity, implies that new data must be gathered. If an entity or attribute is removed, data might also need to be removed. This effort is equal for both versions, since neither of them provide support for the data acquisition stage. Moreover, the *data connector component* needs also to be updated in both versions.

Once these data sets have been updated, we would need to modify the DSL syntax to support these new elements and remove the old ones, as in scenario *ChSc1*. As we mentioned, in FLANDM, this change only impacts the domain entities model, whereas in the initial version, more components would be affected.

5.3.3. ChSc3: customisation of the query transformation process

As previously commented, some parameters of the data mining processes generated during the transformation of a query might need to be adjusted to best fit to a specific domain, with the objective of improving their accuracy. These changes range from the modification of an algorithm parameter of the data mining procedure being instantiated to the creation of a completely new transformation process that employs another algorithms.

In the original version, code generators were monolithic, tangling the abstract transformation process with platform specificities, whereas in FLANDM, these stages are separated, which creates smaller and more cohesive modules. Therefore, this change can be more easily incorporated in FLANDM, since we need only to change some ETL transformations, which are free of target platform details. In the initial version case, these changes implied to search for actual transformation steps inside a code generation template bloated of platform-specific details.

If the change requires the creation of a new data mining procedure, in FLANDM, we would need to add it to the data procedure metamodel. This action would not be required in the initial version, and it is an extra cost we need to pay in FLANDM for achieving a better separation of concerns. Nevertheless, the effort associated to this extra step is low and it usually pays off quickly.

5.3.4. ChSc4: addition of new commands

New analyses might be required in a domain, which implies adding new commands that support them. This change involves updating the DSL syntax, descriptions provided by the content assistant and providing a new full query transformation process. This full transformation process in FLANDM comprises both the query to data procedure transformation step and the code generation step. Although the effort in both cases would be similar, as before, we would benefit of a better separation of concerns in the FLANDM case.

6. Summary and future work

This work has presented FLANDM, a framework designed to reduce the development cost of Data Mining Democratisation Languages (DMDLs), where each language is tailored to a specific domain. Since these DMDLs share several commonalities, the development cost reduction is achieved by means of component reuse and customisation.

This framework was built over a previous work [7], where the architecture of a first DMDL was presented. For FLANDM, we refactored this initial architecture to support and ease component reuse, fixing some flaws detected in it. The refactored infrastructure allows creating DSLs starting from a generic solution, which gets specialized into the final domain through modular changes that are easy to introduce.

Two new elements were included in this refactored architecture: (1) an *entities metamodel*, and (2) a *data procedure metamodel*. The first one contributed to encapsulate domain-specific elements in one module. This helped to make the other components of the architecture domain-independent, and, therefore, more reusable across domains. On the other hand, the *data procedure metamodel* allowed the separation of platform-independent and platform-specific issues, making the code generation components for these DMDLs more cohesive.

The benefits of our approach were evaluated by using a cost model that measured the effectiveness achieved through components reuse. The obtained results show that the development cost of a DMDL can be reduced by 50% thanks to our approach.

As future work, we will consider the two following issues. Firstly, as commented, it is difficult to encapsulate data mining processes that can work accurately for different data sets. So, it is necessary to make changes in these processes when moving to a new domain [66]. Therefore, we will study how some promising techniques, such as *meta-learning* [67] or parameter-less algorithms [27], can help to deal with this issue. In line with this work, we will compare the effectiveness of data mining processes specified with DMDLs against more traditionally-created ones with the help of data mining experts.

Secondly, it can be noticed that this work did not provide help for the domain definition stage, despite the fact that this stage consumes a large portion of the effort of developing a DMDL. To mitigate this problem, we are currently working on a dedicated language, called *Lavoisier* [48], for domain data acquisition.

Acknowledgements

This work has been partially funded by the doctoral studentship program from the University of Cantabria, and by the Spanish Government under grants TIN2014-56158-C4-2-P (M2C2) and TIN2017-86520-C3-3-R.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cl.2018.07.002.

References

[1] Fayyad U, Piatetsky-Shapiro G, Smyth P. From data mining to knowledge discovery in databases. AI Mag 1996;17(3):37–54.
[2] Sweney M. Netflix gathers detailed viewer data to guide its search for the next hit. The Guardian last accessed: 21-02-2018; https://goo.gl/4nQVxE.
[3] Witten IH, Frank E, Hall MA, Pal CJ. Data mining: practical machine learning tools and techniques. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 4th ed.; 2016. ISBN 0128042915, 9780128042915.
[4] Baesens B. Analytics in a big data world: The essential guide to data science and its applications. 1st ed. Wiley Publishing; 2014. ISBN 1118892704, 9781118892701.
[5] Donati G, Woolston C. Information management: data domination. Nature 2017;548:613–14. doi:10.1038/nj7669-613a.
[6] Abadi D, et al. The beckman report on database research. SIGMOD Rec 2014;43(3):61–70. doi:10.1145/2694428.2694441.
[7] de la Vega A, García-Saiz D, Zorrilla M, Sánchez P. Towards a DSL for educational data mining. In: Sierra-Rodríguez JL, et al., editors. Languages, Applications and Technologies SE - 8. Communications in Computer and Information Science, 563. Springer International Publishing; 2015. p. 79–90. doi:10.1007/978-3-319-27653-3_8. ISBN 978-3-319-27652-6.
[8] Kiczales G, des Rivieres J, Bobrow DG. The art of the metaobject protocol. MIT Press; 1991.
[9] Hinkel G, Goldschmidt T, Burger E, Reussner R. Using internal domain-specific languages to inherit tool support and modularity for model transformations. Softw Syst Model 2017. doi:10.1007/s10270-017-0578-9.
[10] de la Vega A, García-Saiz D, Zorrilla M, Sánchez P. A model-driven ecosystem for the definition ofdata mining domain-specific languages. In: Ouhammou Y, Ivanovic M, Abelló A, Bellatreche L, editors. Model and Data Engineering. Cham: Springer International Publishing; 2017a. p. 27–41. doi:10.1007/978-3-319-66854-3_3.
[11] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in software engineering. Springer; 2012. doi:10.1007/978-3-642-29044-2. ISBN 978-3-642-29043-5.
[12] Kitchenham B, Charters S. Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. Keele University and Durham University Joint Report; 2007.
[13] Jalali S, Wohlin C. Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the 2012 6th ACM_IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM); 2012. p. 29–38. doi:10.1145/2372251.2372257. ISBN 9781450310567.
[14] Luna JM, Castro C, Romero C. MDM tool: a data mining framework integrated into Moodle. Comput Appl Eng Educ 2017;25(1):90–102. doi:10.1002/cae.21782.
[15] Jugo I, Kovačić B, Tijan E. Cluster analysis of student activity in a web-based intelligent tutoring system. Pomorstvo: Sci J Marit. Res 2015;29(1):75–83.
[16] Chittaro L, Combi C, Trapasso G. Data mining on temporal data: a visual approach and its clinical application to hemodialysis. J Vis Lang Comput 2003;14(6):591–620. doi:10.1016/j.jvlc.2003.06.003.
[17] Kamsu-Foguem B, Tchuenté-Foguem G, Allart L, Zennir Y, Vilhelm C, Mehdaoui H, et al. User-centered visual analysis using a hybrid reasoning architecture for intensive care units. Decis Support Syst 2012;54(1):496–509. doi:10.1016/j.dss.2012.06.009.
[18] Kamdar MR, Zeginis D, Hasnain A, Decker S, Deus HF. ReVeaLD: a user-driven domain-specific interactive search platform for biomedical research. J Biomed Inf 2014;47:112–30. doi:10.1016/j.jbi.2013.10.001.
[19] Luu TD, Rusu A, Walter V, Linard B, Poidevin L, Ripp R, et al. KD4v: comprehensible knowledge discovery system for missense variant. Nucleic Acids Res 2012;40:W71–5. doi:10.1093/nar/gks474.
[20] Rapidminer. https://rapidminer.com/.
[21] Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, et al. KNIME - the Konstanz Information Miner. SIGKDD Explor Newslett 2009;11(1):26–31. doi:10.1145/1656274.1656280.
[22] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The weka data mining software: an update. SIGKDD Explor Newslett 2009;11(1):10–18. doi:10.1145/1656274.1656278.
[23] Zorrilla M, García-Saiz D. A service-oriented architecture to provide data mining services for non-expert data miners. Decis Support Syst 2013;55:399–411. doi:10.1016/j.dss.2012.05.045.
[24] Ben Ayed M, Ltifi H, Kolski C, Alimi AM. A user-centered approach for the design and implementation of KDD-based DSS: a case study in the healthcare domain. Decis Support Syst 2010;50(1):64–78. doi:10.1016/j.dss.2010.07.003.
[25] Espinosa R, García-Saiz D, Zorrilla M, Zubcoff JJ, Mazón J-N. Enabling non-expert users to apply data mining for bridging the big data divide. In: Accorsi R, Cudre-Mauroux P, Ceravolo P, editors. International Symposium on Data-driven Process Discovery and Analysis, SIMPDA 2013, 203. Springer Verlag; 2015. p. 65–86. doi:10.1007/978-3-662-46436-6_4. 18651348; 9783662464359 (ISBN).
[26] Santos RS, Malheiros SMF, Cavalheiro S, de Oliveira JMP. A data mining system for providing analytical information on brain tumors to public health decision makers. Comput Methods Progr Biomed 2013;109:269–82. doi:10.1016/j.cmpb.2012.10.010.
[27] Balcázar JL. Parameter-free association rule mining with Yacaree. In: Extraction et Gestion des Connaissances (EGC) Brest (France); 2011. p. 251–4.

[28] D. García-Saiz, M.E. Zorrilla, E-learning web miner: a data mining application to help instructors involved in virtual courses, in: Educational Data Mining (EDM), 2011.
[29] Rice W. Moodle 2.0 E-learning course development. Packt Publishing; 2011. ISBN 9781849515269.
[30] Rice W. Blackboard essentials for teachers. Packt Publishing; 2012.
[31] Campos M, Stengard P, Milenova B. Data-centric automated data mining. In: Proceedings of the fourth international conference on machine learning and applications (ICMLA'05); 2005. p. 97–104. doi:10.1109/ICMLA.2005.18. ISBN 0-7695-2495-8.
[32] Reif M, Shafait F, Goldstein M, Breuel T, Dengel A. Automatic classifier selection for non-experts. Pattern Anal Appl 2014;17(1):83–96. doi:10.1007/s10044-012-0280-z.
[33] Ankerst M, Ester M, Kriegel H-P. Towards an effective cooperation of the user and the computer for classification. In: Proceedings of the Sixth ACM SIGKDD international conference on knowledge discovery and data mining; 2000. p. 179–88. doi:10.1145/347090.347124. ISBN 1-58113-233-6.
[34] Kleppe A. Software language engineering: creating domain-specific languages using metamodels. Addison-Wesley Professional; 2008. ISBN 0321553454, 9780321553454.
[35] Strembeck M, Zdun U. An approach for the systematic development of domain-specific languages. Softw: Pract Exp 2009;39(15):1253–92. doi:10.1002/spe.936.
[36] Kosar T, Oliveira N, Mernik M, Varando Pereira MJ, Črepinšek M, Da Cruz D, et al. Comparing general-purpose and domain-specific languages: an empirical study. Comput Sci Inf Syst 2010;7(2):247–64. doi:10.2298/CSIS1002247K.
[37] Völter M, Benz S, Dietrich C, Engelmann B, Helander M, Kats LCL, et al. DSL engineering - designing, implementing and using domain-specific languages. dslbook.org; 2013. ISBN 978-1-4812-1858-0.
[38] Combemale B, France R, Jézéquel J-M, Rumpe B, Steel JR, Vojtisek D. Engineering modeling languages. Chapman and Hall/CRC; 2016. ISBN 9781466583733.
[39] Steinberg D, Budinsky F, Paternostro M, Merks E. EMF: eclipse modeling framework. 2nd ed. Addison-Wesley Professional; 2009. ISBN 0321331885.
[40] Eysholdt M, Behrens H. Xtext: implement your language faster than the quick and dirty way. In: Proceedings of the companion to the 25th annual conference on object-oriented programming, systems, languages, and applications (SPLASH/OOPSLA); 2010. p. 307–9. doi:10.1145/1869542.1869625. ISBN 978-1-4503-0240-1.
[41] Syriani E, Luhunu L, Sahraoui H. Systematic mapping study of template-based code generation. Comput Lang Syst Struct 2018;52:43–62. doi:10.1016/j.cl.2017.11.003.
[42] Paige RF, Kolovos DS, Rose LM, Drivalos N, Polack FAC. The design of a conceptual framework and technical infrastructure for model management language engineering. In: Proceedings of the 2009 14th IEEE international conference on engineering of complex computer systems. IEEE Computer Society; 2009. p. 162–71. doi:10.1109/ICECCS.2009.14. ISBN 978-0-7695-3702-3.
[43] Mernik M, Heering J, Sloane AM. When and how to develop domain-specific languages. ACM Comput Surv 2005;37(4):316–44. doi:10.1145/1118890.1118892.
[44] Smith JW, et al. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In: Proceedings of the Annual Symposium on Computer Application in Medical Care; 1988. p. 261–5.
[45] Kang K, Cohen S, Hess J, Novak W, Peterson A. Feature-oriented domain analysis (foda) feasibility study. Tech. Rep. Pittsburgh, PA: CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University; (1990). URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231.
[46] Čeh I, Črepinšek M, Kosar T, Mernik M. Ontology driven development of domain-specific languages. Comput Sci Inf Syst 2011;8(2):317–42. doi:10.2298/CSIS101231019C.
[47] Evans E. Domain-driven design: Tacking complexity In the heart of software. Addison-Wesley Longman Publishing Co., Inc.; 2003. ISBN 0321125215.
[48] de la Vega A, García-Saiz D, Zorrilla ME, Sánchez P. On the automated transformation of domain models into tabular datasets. In: Proceedings of the ER Forum 2017 and the ER 2017 demo track co-located with the 36th international conference on conceptual modelling (ER 2017), Valencia, Spain, - November 6–9, 2017.; 2017b. p. 100–13.
[49] Fister IJ, Kosar T, Fister I, Mernik M. Easytime++: a case study of incremental domain-specific language development. Inf Technol Control 2013;42(1). doi:10.5755/j01.itc.42.1.1968.
[50] 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. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2000. p. 727–34. ISBN 1-55860-707-2.
[51] Ester M, Kriegel H-P, Sander J, Xu X. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining. AAAI Press; 1996. p. 226–31.
[52] Quinlan JR. C4.5: programs for machine learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1993. ISBN 1-55860-238-0.
[53] Kolovos DS, Paige RF, Polack FA. The epsilon transformation language. In: Proceedings of the 1st international conference on theory and practice of model transformations. Springer-Verlag; 2008. p. 46–60. doi:10.1007/978-3-540-69927-9_4. ISBN 978-3-540-69926-2.
[54] Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Trans Evolut Comput 1997;1(1):67–82. doi:10.1109/4235.585893.
[55] Rose LM, Paige RF, Kolovos DS, Polack FA. The epsilon generation language. In: Proceedings of the 4th european conference on model driven architecture: foundations and applications. Springer-Verlag; 2008. p. 1–16. doi:10.1007/978-3-540-69100-6_1. ISBN 978-3-540-69095-5.
[56] Gaffney J, Durek T. Software reuse - key to enhanced productivity: some quantitative models. Inf Softw Technol 1989;31(5):258–67. doi:10.1016/0950-5849(89)90005-0.
[57] Boehm BW. Improving software productivity. Computer 1987;20(9):43–57. doi:10.1109/MC.1987.1663694.
[58] Kim J, Lee S, Hwang S-W, Kim S. Enriching documents with examples: a corpus mining approach. ACM Trans Inf Syst 2013;31(1):1:1–1:27. doi:10.1145/2414782.2414783.
[59] Johar M, Mookerjee V, Sethi S. Optimal software design reuse policies: a control theoretic approach. Inf Syst Front 2015;17(2):439–53. doi:10.1007/s10796-013-9421-1.
[60] Adams B, Kavanagh R, Hassan AE, German DM. An empirical study of integration activities in distributions of open source software. Empir Softw Eng 2016;21(3):960–1001. doi:10.1007/s10664-015-9371-y.
[61] Tibermacine C, Sadou S, That MTT, Dony C. Software architecture constraint reuse-by-composition. Future Gener Comput Syst 2016;61:37–53. doi:10.1016/j.future.2016.02.006.
[62] Frakes W, Terry C. Software reuse: metrics and models. ACM Comput Surv 1996;28(2):415–35. doi:10.1145/234528.234531.
[63] Rothenberger MA, Dooley KJ, Kulkarni UR, Nada N. Strategies for software reuse: a principal component analysis of reuse practices. IEEE Trans Softw Eng 2003;29(9):825–37. doi:10.1109/TSE.2003.1232287.
[64] Ajila SA, Wu D. Empirical study of the effects of open source adoption on software development economics. J Syst Softw 2007;80(9):1517–29. doi:10.1016/j.jss.2007.01.011. Evaluation and Assessment in Software Engineering.
[65] Deursen AV, Klint P. Little languages: little maintenance? Journal of Software Maintenance: Research and Practice 1998;10(2):75–92. doi:10.1002/(SICI)1096-908X(199803/04)10:2⟨75::AID-SMR168⟩3.0.CO;2-5.
[66] Anand SS, Bell DA, Hughes JG. The role of domain knowledge in data mining. In: Proceedings of the fourth international conference on information and knowledge management. CIKM '95; New York, USA; 1995. p. 37–43. doi:10.1145/221270.221321. ISBN 0-89791-812-6.
[67] Lemke C, Budka M, Gabrys B. Metalearning: a survey of trends and technologies. Artif Intell Rev 2015;44(1):117–30. doi:10.1007/s10462-013-9406-y.