UNIVERSIDAD DE CANTABRIA

Programa de

Doctorado en Ciencia y Tecnología

Tesis Doctoral

Lenguajes Específicos de Dominio para la Democratización de la Minería de Datos

PhD Thesis

Domain-Specific Languages for Data Mining Democratisation

Realizada por

Alfonso de la Vega Ruiz

Dirigida por

Pablo Sánchez Barreiro

Escuela de Doctorado de la Universidad de Cantabria

Santander 2019

A PhD involves failing to do things correctly a lot of times.

My mind tried to sabotage this thesis in several occasions.
Fortunately, I think it also failed at that.

Como dijo una vez una persona más inteligente que yo,
¡RAKUYAKI!

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text, co-authorships in related published articles, and acknowledgements. Some parts of this dissertation appear in peer-reviewed conferences and journals, and in preprint databases. Check the List of Publications on page xxv for details.

Alfonso de la Vega Ruiz
Santander 2019

Agradecimientos

Han sido casi cinco años, y alguno se me ha hecho más largo que otro, pero bueno, finalmente aquí estamos. La verdad es que en algunas ocasiones, siendo sincero, servidor no daba un duro porque llegara este día. Lo que tengo claro es que, sin personas alrededor apoyándome en esos momentos difíciles, no me encontraría en esta mañana escribiendo estos agradecimientos.

Empiezo por Pablo, mi director durante la tesis. Acudí a el proveniente de unos temas de trabajo completamente diferentes a aquellos que finalmente tratamos en esta tesis, y aceptó dirigirme con poco más contacto que haberme dado clase en una asignatura. Tengo que reconocerle que ha sabido manejar a una persona que llegaba con poca tolerancia a la frustración, muy tozuda en cuanto a pensamientos y posturas, y poco acostumbrada a las críticas, aunque éstas fueran constructivas. Creo que, durante esta tesis, hemos conseguido alcanzar un equilibrio en nuestras posturas, que nos ha permitido ser productivos y conseguir la mayor parte de las metas propuestas. Muchas gracias Pablo, y espero que podamos seguir trabajando juntos en el futuro.

Quiero dedicarle también unas palabras a Marta, profesora compañera del departamento. Aunque sobre el papel diga que su labor ha sido ser mi tutora administrativa durante el doctorado, ha superado con creces las obligaciones relativas a esa figura, con avisos de diferentes eventos y congresos, con interesarse por el avance de mi trabajo, o con su consejo cuando he tenido preguntas acerca de cualquier cosa. En definitiva, gracias Marta por toda la ayuda proporcionada.

I would also like to thank Prof. Dimitris Kolovos for accepting me for a research stay. As it was the case with my thesis supervisor, he accepted me to come to York with nothing more than a couple of my initial papers. Although my stay there was too short, it helped me a lot to face the remaining years of my PhD with improved attitude and knowledge. Thanks Dimitris, I hope we can continue collaborating in the future.

Considero tan importante el apoyo de un grupo en cuanto a labores de docencia e investigación, como el valor humano que este grupo transmite, ya sea por su disposición a ayudarte ante diferentes problemas, como por la posibilidad de comentar los resultados de las últimas elecciones (y en esta tesis, ha habido muchas), las criptos, o cualquier

otra discusión filosófico-tecnológica que tocara. Creo que el grupo de Ingeniería del Software y Tiempo Real cumple ambos criterios y con muy buena nota. Quiero dar las gracias a todos los miembros del grupo por permitirme iniciarme en el mundo de la universidad, por apoyarme en los diferentes trámites que he tenido que realizar durante la tesis, o por simplemente charlar conmigo cuando necesitaba desconectar de algún tema en concreto.

Esta tesis me ha permitido conocer y entablar amistad con muchas personas en mi misma o en una situación similar dentro de la universidad. Por el grupo se encuentra Alejandro, con quien he podido comentar muchas cosas acerca de nuevos cacharros tecnológicos y de si se nota un montón o no las ventajas que se supone que ofrecen. Meto por aquí también a Rafa, de reciente incorporación a la facultad, pero ya uno más de nosotros. Igual un día termina comprándose la tele esa. Mención especial merecen mis compañeros de comedor y cafés: David, Andrea, Ujué, y David (también conocido como el gallego, mote a ratos compartido con Óscar, pero con cariño/por colisión de nombres). Termino con Diego, mi compañero de despacho y fatigas, que pertenecía al club del tupper pero luego dejó de molar. Gracias por esas conversaciones psicológico-festivas, que también han contribuido a que esta tesis termine correctamente.

Quiero dar las gracias también a los que han sido mis amigos prácticamente toda la vida, y que espero sigan siéndolo por muchos años. A Manuel, Alberto, Pablo, Marta, Javi, Alejandra, Silvia, Paula, Valeria, Pantor & Ley, y a los que seguro me olvido.

No podía faltar en estos agradecimientos una mención a mi familia. Muchas horas he dedicado a la tesis dejando de lado a la gente que más me importa, y uno de mis objetivos actuales es recuperar ese tiempo y evitar que vuelva a suceder. A Marcos y a Sara, por estar siempre ahí cuando lo he necesitado, y por supuesto sin pedir nada a cambio; y a Adela, que con lo trisca que es su tío (y lo que te queda maja), espero que no me acabe cogiendo tirria. Finalmente, a mis padres, Alfonso y Maria Rosa, que pese a la impotencia que sentían a veces al verme desanimado en estos años, nunca dejaron de ayudarme con todo lo que pudieron para que siguiera adelante. No creo que sea capaz de devolveros todo lo que habéis hecho por mí, pero lo intentaré lo mejor que pueda.

Termino estos agradecimientos con la auténtica responsable de que esta tesis haya llegado a buen puerto. Una de las únicas personas capaces de aguantarme a mí y a mis neuras durante todos estos años, principalmente a base de collejas cuanto mi mente se encasquillaba con algo. Calculo que me sería imposible encontrar a alguien igual en esta y otras tantas vidas. Muchas gracias Yael, ahora te toca a ti (manosescribiendoenelaire.gif).

Abstract

Currently, computer systems gather large amounts of data that, when properly analysed, can be of great help for different purposes. For instance, data mining techniques can be used to discover insights previously hidden in data.

Nevertheless, the correct usage of these techniques requires sound knowledge in very specialised concepts, such as analysis algorithms, advanced statistics, or data management. People willing to analyse data often lack this knowledge, which hampers *data mining democratisation*.

In this thesis, we explored whether *Model-Driven Engineering (MDE)* and *Domain-Specific Languages (DSLs)* technologies can be of help to the data mining democratisation field. These technologies have demonstrated their effectiveness to provide domain-adapted solutions, which are easy to use and feel familiar to experts in an application domain. Therefore, the use of these technologies could contribute positively to the goal of enlarging the spectrum of people that can apply data mining techniques.

We started our work by reviewing the state of the art of this field, which allowed us to identify any shortcomings of the existing approaches. Some of these shortcomings where: (1) those analysis solutions that are completely domain-independent may exhibit accuracy problems, as they do not take into account the specificities of the domain to configure the analysis processes; and (2) the task of easing the selection and preparation of the data that are to be used in an analysis has been scarcely addressed in the literature.

From our findings in this review, we devised several approaches to provide non-expert users with different DSLs for data mining democratisation. Namely, we developed FLANDM, i.e., a model-driven framework for the rapid generation of DSLs for data mining adapted to the specificities of each concrete context. DSLs generated with this framework provide non-experts with a query-based syntax to invoke analysis processes. Queries are formulated by combining high-level commands with terminology from the domain, hiding any technical details of the executed processes to the end user.

Additionally, this framework uses two DSLs, *Lavoisier* and *Pinset*, which are in charge of making data conform to the requirements imposed by the executed data mining algorithms. Lavoisier offers an easy-to-use syntax to non-experts for selecting the information to analyse from a domain model, which acts as a high-level representation of the available data. Complementary, Pinset provides advanced users with powerful syntax constructs to perform more advanced data computations, which require a lower-level control of the data transformation process.

We also describe how we validated our approaches, e.g., by generating DSLs for several domains. Moreover, we performed a set of empirical experiments to state whether the DSLs generated with FLANDM might be actually used by people without knowledge on data mining techniques. The results of these experiments show that, after receiving a minimum training, most non-expert users could employ the generated DSLs to invoke data mining processes over data from their domain.

Resumen

En la actualidad, los sistemas informáticos recogen grandes cantidades de datos que, si se analizan adecuadamente, pueden resultar de gran ayuda para diferentes fines. Por ejemplo, las técnicas de minería de datos se pueden utilizar para descubrir patrones en datos en principio no detectables a simple vista.

Sin embargo, el uso correcto de estas técnicas requiere del conocimiento de ciertos conceptos muy especializados, como algoritmos de análisis, estadísticas avanzadas o gestión y tratamiento de datos. Gran parte de las personas que disponen de datos que les gustaría analizar a menudo carecen de este conocimiento, lo que dificulta la democratización de la minería de datos.

En esta tesis analizamos si la Ingeniería Dirigida por Modelos (MDE) y los Lenguajes Específicos de Dominio (DSLs) pueden resultar de ayuda para conseguir una efectiva democratización de la minería de datos. Estas tecnologías han demostrado su eficacia para proporcionar soluciones adaptadas a cada contexto, fáciles de usar y que resultan familiares para los expertos en un dominio de aplicación. Por lo tanto, el uso de estas tecnologías podría contribuir positivamente al objetivo de ampliar el número de personas que pueden aplicar técnicas de minería de datos.

Nuestro trabajo comenzó con una revisión sistemática del estado del arte en este campo, lo que nos permitió identificar cualquier aspecto a mejorar en los enfoques existentes. Algunos de estos aspectos fueron: (1) aquellas soluciones para facilitar el análisis que son completamente independientes del dominio de aplicación pueden presentar problemas de precisión en sus resultados, ya que no tienen en cuenta las especificidades del dominio para configurar los procesos de análisis; y (2) la tarea de facilitar la selección y preparación de los datos que se van a utilizar en un análisis apenas se ha abordado en la literatura.

A partir de los resultados de esta revisión, nuestro trabajo consistió en el diseño de distintas contribuciones para para proporcionar a los usuarios no expertos diferentes DSLs para la democratización de la minería de datos. Una de estas contribuciones es FLANDM: un entorno de desarrollo basado en modelos para la rápida generación de DSLs que permiten realizar análisis de datos adaptados a las especificidades de cada contexto concreto. Los DSLs generados con este framework proporcionan a los no expertos una sintaxis basada en consultas para invocar procesos de análisis. Estas consultas se formulan combinando comandos de alto nivel con terminología del dominio, ocultando al usuario final cualquier detalle técnico de los procesos ejecutados.

Adicionalmente, este framework utiliza dos DSLs, *Lavoisier* y *Pinset*, que se encargan del proceso de transformar y preparar conjuntos de datos a analizar para que

cumplan con los requisitos impuestos por los algoritmos de minería de datos ejecutados. Lavoisier ofrece una sintaxis usable por no expertos para seleccionar la información a analizar a partir de un modelo de dominio, que actúa como una representación de alto nivel de los datos disponibles. De forma complementaria, Pinset proporciona a usuarios avanzados con conocimientos en programación una serie de construcciones sintácticas potentes para realizar cálculos de datos más complejos, que requieren un control más detallado del proceso de transformación de datos.

Durante nuestra investigación realizamos diferentes evaluaciones para valorar nuestras contribuciones, por ejemplo, mediante la generación de DSLs para varios dominios. Además, llevamos a cabo una serie de experimentos empíricos para determinar si los DSLs generados con FLANDM podrían ser realmente utilizables por personas sin conocimientos sobre técnicas de minería de datos. Los resultados de estos experimentos muestran que, tras recibir una formación mínima, la mayoría de estos usuarios fueron capaces de emplear el DSL proporcionado para invocar procesos de minería de datos sobre los datos de su dominio.

Contents

List of Figures

List of Tables

List of Publications

The work of this thesis has been peer-reviewed in several conferences and journals. A list of publications in inverse chronological order containing the contributions of this thesis can be found below.

- (To be published) de la Vega, and Sánchez, P. (2019). Model-Driven Technologies for Data Mining Democratisation. STAF 2019 Junior Researchers Community Event, CEUR Workshop Proceedings.
- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2019). How Far are we from Data Mining Democratisation? A Systematic Review. arXiv e-prints 1903.08431 (2019), https://arxiv.org/abs/1903.08431
- de la Vega, A., Sánchez, P., and Kolovos, D. (2018). Pinset: A DSL for Extracting Datasets from Models for Data Mining-Based Quality Analysis. 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 83–91. https://doi.org/10.1109/quatic.2018.00021
- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2018). FLANDM: a development framework of domain-specific languages for data mining democratisation. Computer Languages, Systems and Structures, 54, 316–336. https://doi.org/10.1016/j.cl.2018.07.002
- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2017). On the Automated Transformation of Domain Models into Tabular Datasets. ER FORUM, CEUR Workshop Proceedings, 1979. Retrieved from http://ceur-ws.org/Vol-1979/paper-07.pdf
- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2017). A Model-Driven Ecosystem for the Definition of Data Mining Domain-Specific Languages. In Model and Data Engineering (Vol. 9893, pp. 27–41). https://doi.org/10.1007/978-3-319-66854-3_3

- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2016). Desarrollo Eficiente de Lenguajes Específicos de Dominio para la Ejecución de Procesos de Minería de Datos. Jornadas de Ingeniería Del Software y Bases de Datos (JISBD). http://hdl.handle.net/11705/JISBD/2016/019

- de la Vega, A., García-Saiz, D., Zorrilla, M., and Sánchez, P. (2015). Towards a DSL for Educational Data Mining. In Languages, Applications and Technologies (pp. 79–90). https://doi.org/10.1007/978-3-319-27653-3_8

Acronyms

ANTLR	ANother Tool for Language Recognition
ATL	Atlas Transformation Language
CSV	Comma-Separated Values
DMDL	Data Mining Democratisation Language
DSL	Domain-Specific Language
EBNF	Extended Backus-Naur Format
EDM	Educational Data Mining
EGL	Epsilon Generation Language
EMF	Eclipse Modeling Framework
ETL	Epsilon Transformation Language
FLANDM	Framework to develop LANguages for Data Mining
GPL	General Purpose Language
KPI	Key Performance Indicator
LOC	Lines of Code
M2M	Model-to-Model
M2T	Model-to-Text
MDE	Model-Driven Engineering
MOF	Meta-Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
SSBI	Self-Service Business Intelligence
UML	Unified Modeling Language

Chapter 1

Preliminaries and Objectives

1.1 Introduction

We live in a time where huge amounts of data are gathered and stored by computer systems. The analysis of these data can be very beneficial for the success of an organisation or project (Caro-Gutiérrez et al. [31], Li et al. [105], Márquez-Vera et al. [120]). For instance, there are companies that base their business model in exploiting information-rich data. We comment here on three of these companies: *Uber*, *Netflix* and *Yelp*.

Uber[1] connects, through a software system, particular drivers offering transport services with people who need to move around cities. The information that this system gathers has been used to identify travel habits in the cities where it operates. This information has been found of great value for the purposes of improving the existing public transport services. Several city councils have acknowledged this by purchasing access to this information (Dungca [51]).

Netflix[2] offers media contents through an online service. Part of this content is produced by Netflix itself. To help determine what media content should be produced, data scientists at Netflix study the activity habits of their clients, which allows creating those kinds of shows that seem to have potential to be successful (Sweney [154]).

Yelp[3] offers a business review service, where owners can advertise their businesses, and customers can give their opinion about them. Yelp makes use of the data it stores to connect owners and potential customers, e.g., by recommending new businesses to users based on the previous activity of these users.

[1] https://www.uber.com/
[2] https://www.netflix.com/
[3] https://www.yelp.com/

In summary, nowadays data are gathered about almost any aspect of our lives, and the analysis of these data can help take better decisions and improve systems and organisations. This analysis is often performed through the use of data mining techniques (Han et al. [71], Witten et al. [167]), which allow discovering useful insights not previously visible in the original data.

Nevertheless, to employ these techniques for analysing data, proficiency is required in different skills, e.g., analysis algorithms, advanced statistics and data transformation processes. Unfortunately, decision makers willing to analyse a data bundle, e.g., city planners or film producers, often lack these skills. As such, they have to rely on experts, i.e., data scientists, to perform this analysis. Since data scientists are a scarce resource (Donati and Woolston [50]), relying on them implies a considerable cost and extra delays.

To alleviate this issue, the *Data Mining Democratisation* field (Cao [30], de la Vega et al. [45]) aims to make data mining techniques more usable by people without an advanced knowledge in them. Existing works in this field try to help achieve this goal through different approaches. For instance, some works try to automate certain tasks of a data mining process, while others focus on the development of easy-to-use solutions that allow invoking analysis processes using high-level interfaces or constructs, which hide low-level details of the executed processes to the end users.

With relation to the latter, *Model-Driven Engineering (MDE)* (Brambilla et al. [24]) and *Domain-Specific Languages (DSLs)* (Kleppe [94], Völter et al. [163]) have demonstrated to be effective methods to provide domain-adapted solutions that are easy to use and feel familiar to experts in an application domain. Examples of successful applications of MDE techniques can be found in different domains, such as the automotive (Chen et al. [32], Zolotas et al. [175]), aerospace engineering (Trase and Fink [158]), embedded software development (Voelter et al. [162]), data communications (Baker et al. [11]), or web engineering (Martínez et al. [113]). The reader interested in the adoption of MDE at the industrial level can refer to wider surveys, like the ones provided by Hutchinson et al. [77] and Ameller and et al. [6].

In this thesis, we explored whether the benefits of MDE technologies can contribute to the democratisation of data mining. Precisely, we worked on the definition of domain-specific languages for aiding non-expert users to analyse data. These languages provide high-level constructs that hide technical details of a data mining process, so that the languages can be employed by people without experience in data mining techniques.

We started our work by performing a literature review of the data mining democratisation field. As a result of this study, we concluded that those approaches that are generic, i.e., they are completely domain-independent, might exhibit accuracy problems, because they do not take into account the particularities of each domain to configure data mining algorithms or to preprocess input data. We also found some interesting approaches that try to alleviate this shortcoming by providing an initially generic analysis framework that can be adapted to each concrete domain to avoid the problems manifested by domain-independent approaches.

In this context, we considered that the DSL engineering paradigm might help provide this required domain adaptations. To perform a more in-depth exploration of this idea, we devised a data mining DSL for the analysis of data coming from the educational domain. Our goal was that this language could be directly used by teachers. With the assistance of this language, teachers would be able to invoke data mining processes using high-level syntax constructs, as well as vocabulary from their domain. These constructs would be then translated through MDE-based technologies into an analysis process, which is executed in the background. The end user is completely agnostic of any low-level details of the executed process, as she only receives the results of such a process.

The development of this first DSL for data mining revealed that this kind of DSLs might be too expensive to develop from scratch in most cases. To mitigate this cost, we designed *FLANDM (Framework to develop LANguages for Data Mining)*, an MDE framework that can be used to generate DSLs for data mining. Our evaluation of the cost-reduction capabilities of this framework showed that the cost of generating these DSLs gets reduced by about 50%.

To evaluate the effectiveness of the generated languages, we performed empirical studies with an analysis language for the educational domain. In these experiments, teachers without experience in data mining techniques executed analysis processes over data from university courses. The results of these experiments showed that, after a minimum training, most teachers were able to execute different data mining processes over the provided data.

Other important issue identified during our literature review is that the selection and preparation of the data to be used in an analysis is a process scarcely addressed in the literature. To alleviate this shortcoming, we developed two languages, called *Lavoisier* and *Pinset*. Each language was designed to allow the mentioned data selection and preparation steps with a different kind of end user in mind. Lavoisier can be used by people without any data management skills, as it offers an easy-to-use syntax to

perform data selection over a high-level representation of the available data. On the other hand, Pinset is more oriented for people with programming skills that can take advantage of these to have a more fine-grained control of the data preparation process, e.g., by performing some advanced operations that are not supported in Lavoisier.

The rest of this chapter is structured as follows: Section 1.2 describes some analysis case studies that we used in this thesis work. In Section 1.3, we give an overview on the parts of a data mining process. Section 1.4 enumerates some of the challenges that need to be confronted when trying to democratise data mining. Section 1.5 describes our motivation to develop DSLs for democratising data mining. Section 1.6 gives a background on MDE technologies and DSLs engineering, which might be skipped by a reader with experience in these fields. In Section 1.7, we enumerate the resulting contributions of this thesis. Finally, Section 1.8 describes the content of the remaining chapters of this document.

1.2 Motivating Analysis Examples

This section presents some of the case studies that motivated the development of this thesis. We use these case studies to illustrate our contributions throughout this dissertation.

1.2.1 E-learning Platforms

Nowadays there are a lot of courses that are offered through, or use, online web-based systems, which are known as *e-learning platforms*. *BlackBoard* [139] or *Moodle* [138] are two examples of these platforms. From the webpage of a course, students can watch video lectures, access learning material, submit assignments, ask questions in a forum or perform tests, among other tasks. Depending on the nature of the course, it can be completely online, or the activities in the e-learning platform can be complemented with in-person classroom lessons. The platforms that host the courses log data of the students activity, such as the use of the learning material, how do students navigate the site, which students visit and participate in the forum, and so on.

Educational Data Mining (EDM) (Romero and Ventura [140]) is a field that aims to take advantage of the data gathered by e-learning platforms. Romero and Ventura define this field as *"an emerging discipline, concerned with developing methods for exploring the unique types of data that come from educational settings, and using those methods to better understand students, and the settings which they learn in."* [140].

The discovered information could be useful for teachers and instructors to improve the performance of their teaching processes.

For instance, at the beginning of a course, a teacher could ask: *"What are the different types of students in my class?"*. This information can be computed by using *clustering techniques* [71] over the students' demographic data, e.g., age, rural or metropolitan origin; or success in previous courses. Using this information, the teacher might adapt the course before it starts to fine-tune it according to their students' particularities.

When the course finishes, teachers are usually interested in the students that have not passed. Therefore, they would like to refine the previous question and ask: *"What are the types of students that have not passed?"*. As before, this information can be computed using clustering techniques on the students' demographic data, but now it is also possible to include the activity data gathered by the e-learning platform. These data allow obtaining different indicators, such as the number of online sessions in the course webpage, the average duration of these sessions, the number of messages written in the course forum, or the use of the learning materials provided in the webpage.

Moreover, teachers are obviously interested in asking *"What are the reasons why some of my students failed?"*. This question might be partially answered by applying *classification techniques* [71] on the students' data, by analysing the student activity logs to find out these reasons.

1.2.2 Business Recommendation Systems

Yelp is an American company that provides an online business review service. In this service, owners can describe and advertise their businesses and customers can write their opinions about these businesses.

For each registered business, Yelp provides information about its location, the different features it offers, like the availability of Wi-Fi or a smoking area, and the categories that best describe it, e.g., Cafes, Restaurant, Italian, Gluten-Free, and so on. Users can make reviews of these businesses, rate them and introduce a text describing their experience. Additionally, users can write *tips*, which are small pieces of advice about a business, such as *do not miss its salmon!* Yelp also provides some social network capabilities, so users can have *friends* or *fans*, and they can receive *votes* in their reviews in case other users found these reviews *funny*, *useful* or *cool*.

These data could be used to study different questions business owners may have. For instance, an analysis could be performed to identify the main qualities of those business that are successful, e.g., a good location, opening times, availability of WiFi

and/or smoking areas, notices of food restrictions, or the importance of having private parking. Another analysis could involve studying the text reviews given by clients to detect pleased or angry comments (sentiment analysis).

1.2.3 Artefacts from Software Projects

Data mining techniques are being employed to improve different aspects of software quality assurance processes (Babur et al. [9], Beller et al. [19], D'Ambros et al. [40], Di Rocco et al. [48], F. Palomba et al. [56], M. Ochodek et al. [109], Malhotra [111]). Among other issues, these techniques have been used to: (1) predict the existence of software *bugs* [111]; (2) detect patterns or *smells* that might affect software quality [56]; or (3) obtain intelligent metrics that provide better insights for quality analysis [109], among others. These techniques are applied to different kinds of software artefacts, e.g., source code [40], test reports [19], or software models [9, 48].

In general, the objective of these approaches is to develop prediction models, which can help in the automatic detection of complex or hidden issues that might affect the quality of a software product. These prediction models are constructed by different algorithms, which use information extracted from existing software repositories and historical records of previous software projects.

As an example, D'Ambros et al. [40] gathered and made publicly available metrics and historical data about five open source software systems. They used these data to train fault-detection predictors for software products.

1.2.4 Test Results from Clinical Patients

Data mining is used extensively in the medical field (Chittaro et al. [34], Kamsu-Foguem et al. [86], Klenk et al. [92], Smith and et al. [150]). For instance, Smith and et al. [150] analysed information gathered from a group of patients that were tested for diabetes. 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. Other possible analyses could involve the classification of patients in groups according to some features, or the ranking of the most relevant indicators when determining if a patient has or does not have diabetes.

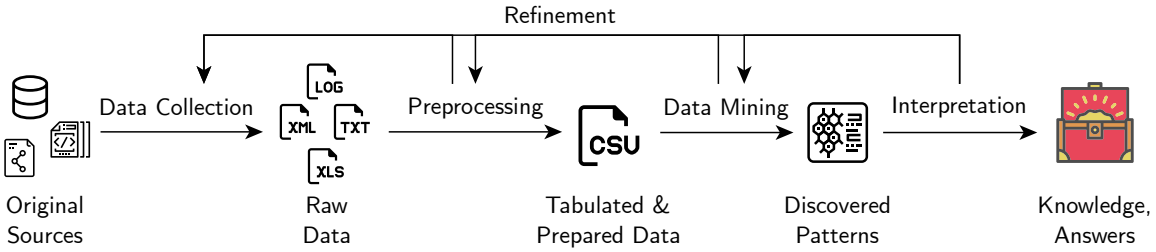


Fig. 1.1 Stages that conform a data mining process.

1.3 Data Mining Processes

Data mining processes are performed to seek for answers to business questions using any available bundle of data (Witten et al. [167]). Every analysis must start with a thorough comprehension about the business domain and the concrete questions to answer. For instance, a business question in the context of software development projects as described in Section 1.2.3 could be: *What common characteristics are shared between the classes of my object-oriented projects that contain bugs?* Additionally, existing data sources and the meaning of these data in the concrete domain must be identified. A negligence in detecting important factors might incur extra expenses, in the form of bad decisions taken or lost time and effort.

Using data mining techniques is not a monolithic process, but rather a chain of different stages (U. Fayyad et al. [159], Wirth [166]). Figure 1.1 shows a diagram of these stages, which can be summarized as follows:

Data Collection The information to analyse has to be collected from its original sources. In some cases, it may be interesting to include heterogeneous sources of information, such as databases, activity log systems, or real-time data streams. Moreover, we might need to retrieve data of different types, from structured data to text documents or even media files. As a result, the collection techniques might vary drastically depending on the input types, ranging from simple SQL queries to invoking advanced operations for extracting data from a web service API.

Preprocessing The obtained raw data are, in most cases, unusable for an analysis (Han et al. [71, see Chapter 3]). These data might contain incomplete or inconsistent information, which must be cleaned and integrated into an appropriate format before being used. For instance, some missing values in data might need to be filled in certain cases; or it could be necessary to normalise numerical data into a [0, 1] interval.

In addition, it might happen that not every piece of initially collected data is relevant for an analysis. Therefore, an exploration and selection step is required to only work with the interesting data fragments for every kind of analysis.

Data Mining This is the stage where the analysis itself takes place. Different techniques might be applied depending on the final goal. For example, if we wanted to group data items according to their similarity, a *clustering algorithm* may be executed. If the objective is to predict future outcomes of a variable, a *classification* or *regression* technique might be more adequate. Apart from using the appropriate analysis technique, we have to select one algorithm from the multiple alternatives available for each technique. For instance, for classification purposes, we might use decision trees, rule-based algorithms, neural networks, k-nearest neighbours, or Bayesian systems, among others. Moreover, not only it is important to select an adequate algorithm to employ: most data mining algorithms require the configuration of several parameters that might considerably affect its performance.

As an example, the C4.5 algorithm (Quinlan [135]) builds a decision tree for the classification of instances from a dataset. This algorithm offers different configurable parameters, such as the minimum number of instances in a leaf, deciding whether to only use binary splits when creating branches, or applying a post-processing pruning to the resulting tree. These parameters determine the depth and width of the resulting tree, and can help improve (or, if badly set, deteriorate) the accuracy of the classification results (Lin and Chen [106]).

Interpretation Once the analysis algorithms are executed, the returned results must be evaluated for soundness and quality. For instance, the accuracy of prediction models is usually evaluated with techniques such as *hold-out* or *cross-validation* (Han et al. [71, see Chapter 8.5]). These techniques can be used to determine whether prediction algorithms work properly with the input data.

The results of an analysis are usually represented with some kind of visualisation to ease their interpretation. For example, the characteristics of each group identified by a clustering algorithm might be represented using a radar chart, also known as Kiviat diagrams. However, if these visualisations are poor, they may make the results difficult to understand. Thus, it is important to pay some attention to any generated visualisations or reports, in order to maximize comprehensibility.

Refinement Frequently, new issues arise when performing the steps of an analysis. For instance, mistakes or wrong assumptions made during the process might be found, e.g., students taking several days to complete a 30-minutes test. Also, any new insights discovered from the results of an algorithm might drive decision makers into new questions, e.g., a teacher finds out that those students coming from urban cities have better grades, and wants to refine her analysis to study this phenomenon. This is why data mining is usually defined as a cyclic and iteratively-refined process (Wirth [166]), because sometimes it is required to come back to a previous stage to fine-tune some settings before continuing with successive tasks.

A lot of data mining products exist without being noticed by final users, as these products operate passively and are included in people day-to-day utilities, such as recommendation systems of e-commerce applications. On the other hand, there are scenarios where users are willing to perform the mining processes actively. Continuing with the online commerce example, a product manager may be interested in knowing the sales trend for the following term; or the profile of those clients that purchased a concrete item.

Unfortunately, most users who want to proactively analyse data lack the required knowledge to perform the data mining process described above. Thus, some authors have identified a gap between data mining techniques and the people who want to employ them (Cao [29], Schlesinger and Rahman [146]). In the last years, several researchers have tried to democratise data mining techniques by filling this gap (e.g. Cao [30], Reif et al. [137], Zorrilla and García-Saiz [176], among others). Next section presents some of the challenges that must be overcome to achieve data mining democratisation.

1.4 General Democratisation Challenges

Each stage of a data mining process described in the previous section requires proficiency in some technical issues that might represent a challenge for *average decision makers*, these are, experts in their respective domains but that have no expertise in data mining techniques. In this section, we enumerate these challenges, and identify what approaches could be developed with the objective of tackling them.

1.4.1 Preparation of Data

The vast amount of data types and storage systems make the collection and processing of data one of the hardest and more time-consuming operations of an analysis process (Crone et al. [38], Munson [119]). The preparation of data involves a series of subtasks, which we comment below. These subtasks are not necessarily performed in the same order we describe them.

1. **Data Extraction**. This task requires different skills depending on the data source. For instance, the SQL language is the de facto option to extract data from a relational database. Although the syntax of SQL tries to resemble expressions from natural languages (e.g. *"SELECT name FROM employees WITH salary > 25000"*), queries generated with this language can get complex very fast, due to the complexity of some of its constructs (aggregation queries, window functions, management of dates, among others). In addition, more complex techniques such as *web scraping* or *data warehouse manipulation* might be required to extract interesting data for an analysis.

2. **Data Cleaning**. Once extracted, data have to be cleaned to detect any errors in the source data, noise in the form of outliers, or to treat missing values. For instance, when working with data about people, we may find that an *age* field includes as values zero years, a thousand, or even a negative number. Because of the different errors that might need to be fixed, this process is mostly manual, and requires the use of low-level tools and data management libraries such as Pandas (McKinney [114]).

3. **Data Integration**. For data coming from different sources, an integration phase must also be performed, which can be a tedious process. For instance, the multiple services of an e-commerce platform might store the same products with similar but slightly different names, which might provoke inconsistencies when merging data.

4. **Data Formatting**. Lastly, it has to be determined which subset of the cleaned data will be used to answer each business question. Once selected, this subset has to be formatted according to the peculiarities of the algorithm to be executed. For example, most clustering algorithms only work with numerical data, so any categorical fields such as sex or marital status have to be removed or translated to an appropriate numerical representation.

As it can be seen, these subtasks demand some specialised skills that average decision makers often lack. So, it would be desirable to create high-level tools or languages for the assistance of non-expert people during this stage of the process.

1.4.2 Algorithm Selection

There are a plethora of data mining algorithms available, each one with their strengths and weaknesses (Wolpert [171]). As an example, to group data in different clusters, one could use a centroids-based solution such as k-means (MacQueen [110]); a hierharchical clustering algorithm (Müllner [121]); or a probabilistic algorithm such as *Expectation Maximization (EM)* (Dempster et al. [46]), among others.

As before, average decision makers are expected to have the knowledge required to select one of these algorithms. So, a solution to assist non-experts when choosing a suitable algorithm for an analysis could help improve the situation of this issue. Even more, if this selection could be automated, non-experts would have to worry about one less thing.

1.4.3 Algorithm Configuration

Each algorithm has its own peculiarities, and custom parameters that must be set before performing an analysis. As an example, in the previous section we enumerated some parameters of the C4.5 decision tree classifier (Quinlan [135]). These parameters affect how the tree is generated, and thus its prediction performance (Lin and Chen [106]). A knowledge of the effect of each one of these parameters is required for the proper usage of the algorithm.

Again, decision makers might lack these technical skills. One option to alleviate this issue could be the definition of a tool to automatically configure these parameters for a given problem, or the existence of auto-configurable, parameter-less algorithms (Zorrilla et al. [177]).

1.4.4 Accidental Complexity of Data Mining Tools

In software, *Accidental Complexity* refers to the difficulties that are caused by a software system, which are not inherent to the concrete task that the system is helping end users accomplish.

With this concept in mind, data mining tools and algorithms have been designed by data scientists, for data scientists. These data scientists are technical-savvy in aspects

such as programming or managing data. So, a large part of existing analysis tools are only available as software libraries, such as the Python Pandas library for data processing [114]; or as fully-fledged programming languages for data analysis, such as R [157]. The skills required to employ these tools are usually too advanced for the average decision maker. Even so, data scientists are often only proficient in a subset of the plethora of analysis tools that are available, and the inclusion of a new one into their toolset is not an immediate process. The definition of user-friendly interfaces or wizards to assist non-experts could alleviate the entry barrier caused by the accidental complexity of some of these tools.

1.5 DSLs for Data Mining Democratisation

The previous section showed that people willing to analyse data must face a lot of technical details related to data mining techniques. These details can quickly become an unconquerable challenge. A possible solution to avoid this complexity could be to hide these technical details behind a high-level interface, so decision makers would deal only with issues belonging to their domain of expertise. This solution could allow non-expert users to employ data mining techniques without having to know about the low-level details of the analysis processes taking place.

Pursuing a similar objective, Model-Driven Engineering technologies have been employed for increasing the level of abstraction at which software systems are developed, enabling most domain experts without programming skills to participate in this process (Chen et al. [32], Trase and Fink [158], Voelter et al. [162], Zolotas et al. [175]). These technologies provide end users with domain-adapted solutions, which include concepts and vocabulary of the context where they are deployed. This adaptation makes these solutions feel familiar and easier to use for the experts of a domain. As an example, Capella[4] is a model-based software engineering tool that allows designing the system, software and hardware architecture of an engineering project or application.

Domain-Specific Languages (DSLs) are one of these model-driven technologies (Kleppe [94], Völter et al. [163]). These languages offer a reduced syntax, as compared to General-Purpose Languages (GPLs) such as Java or C++. On the other hand, this syntax has the advantage of being tailored for performing a specific task or set of tasks in a concrete domain. For instance, the SQL language mentioned in previous sections is a DSL focused in the extraction and management of data contained in relational databases.

[4] https://www.polarsys.org/capella/

Syntax tailoring provides the following benefits:

1. DSLs are very *concise*, in the sense that their syntax constructs offer a great expressivity with a reduced set of information being typed out.

2. As the DSLs are concise and expressive, users can define ideas and objectives *faster* than with conventional GPLs.

3. The syntax of these DSLs is *adapted to the domain* where they are used, e.g., SQL's syntax is prepared to deal with tables, columns, foreign keys, and so on. So, the terminology and vocabulary appearing in a DSL is already known by decision makers, as it contains terms that they use daily.

The enumerated benefits seek making DSLs easy to use for domain experts. These benefits motivated us to investigate the following research hypothesis:

> *Model-driven technologies and domain-specific languages could contribute to an effective data mining democratisation.*

As a result of studying this hypothesis, we devised different MDE and DSL-based contributions for data mining democratisation. Before we delve in the description of these contributions, a background of Model-Driven and Language Engineering concepts is presented in the next section.

1.6 Background

The contributions of this thesis exploit the benefits offered by model-driven and DSL technologies. To make this dissertation self-contained, we provide some background on the fundamental concepts of these technologies. The experienced reader might skip this section.

First, we describe the general concepts behind the MDE and the DSL engineering field, and then we describe how domain-specific languages can be defined following a model-driven approach, which is the one we applied during this work.

1.6.1 Model-Driven Engineering

A *model* can be defined as a representation of one or several elements from a concrete domain, and at a level of abstraction of interest. Fields such as physics, biology, economy, or most engineering disciplines employ models extensively, e.g., to simulate

experiments before their (probably expensive) execution in a laboratory; or to check the design blueprints of a car or a building, with the objective of assessing their integrity and avoiding as many later problems as possible.

According to Stachowiak [151], a model needs to have three features:

- **Mapping Feature**: A model is based on an original, which might not yet exist.
- **Reduction Feature**: A model only represents a (relevant) selection of the properties of an original.
- **Pragmatic feature**: A model needs to be usable in place of an original with respect to some purpose.

Models can be involved in software engineering processes at different levels of implication. For instance, engineers might draw informal models on a whiteboard to discuss a concrete aspect of a system. If these models were considered more important for the project, they could be cleaned up and organised with many others to serve as documentation of the developed system. Nevertheless, these models would be a separate representation of the system codebase, and as such they might end up outdated if future changes are introduced only in the source code of the system. The need to maintain these models in parallel to the code hinders their usage along the life-cycle of a software project.

To improve this situation, *Model-Driven Engineering (MDE)* (Brambilla et al. [24]) aims to promote models as first-class citizens of the software engineering process. Following this approach, models are not only used for descriptive and communicative purposes, but they are also used to automatically generate other models, specific parts of a software system or, in some cases, even the whole software system. Some benefits of this approach include:

1. The automation of repetitive tasks that are prone to errors (Kleppe et al. [95]).
2. The ability to work at a higher level of abstraction, avoiding concrete technology details until necessary (Kleppe et al. [95], Selic [148]).
3. The opportunity of verifying the correctness of huge systems more easily, just like it happens in other disciplines such as physics, biology or civil engineering (Doldi [49]).

In software development processes following a model-driven approach, several types of models can be used to represent the same system at different levels of abstraction.

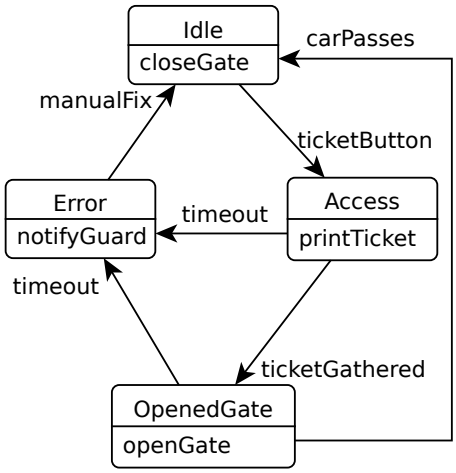


Fig. 1.2 State machine of a parking gate.

Lowering the abstraction level implies moving to a more fine-grained and detailed model, and vice versa.

For instance, lets consider a state machine defined to represent the simplified behaviour of a parking gate entrance system. A graphical model of such a state machine can be found in Figure 1.2. Rectangles with rounded corners represent the *states* of the gate system, these are, *Idle*, *Access*, *OpenedGate*, and *Error*. These states are interconnected with arrows that represent the *events* firing *transitions* from an origin state to a target one. The statements appearing below the name of each state are the *commands* that would be executed when entering the state. For instance, when entering the *OpenedGate* state, the *openGate* command is invoked.

The behaviour of this gate system would be as follows. The system stays *Idle* until a client pushes the *ticket button*, triggering the corresponding event. This event provokes the system to reach the *Access* state, where a ticket that the client has to gather is *printed*. Once gathered, the system transitions to the *OpenedGate* state, which opens the gate, and waits for the car to enter the parking. When the system detects that the *car has passed*, it returns to the initial *Idle* state, which *closes the gate*. If any problems are detected in some of the states (e.g. a client takes too much time to pick up a ticket, or to cross the open gate), a *timeout* event is triggered to notify the parking guard of the existence of some kind of *Error*. This guard is responsible then of *manually fixing* this problem.

From the described state machine model, we could be interested in obtaining executable code expressed in a programming language, such as Java. This code could be used to program the gate system of a real parking. It is possible to obtain this code

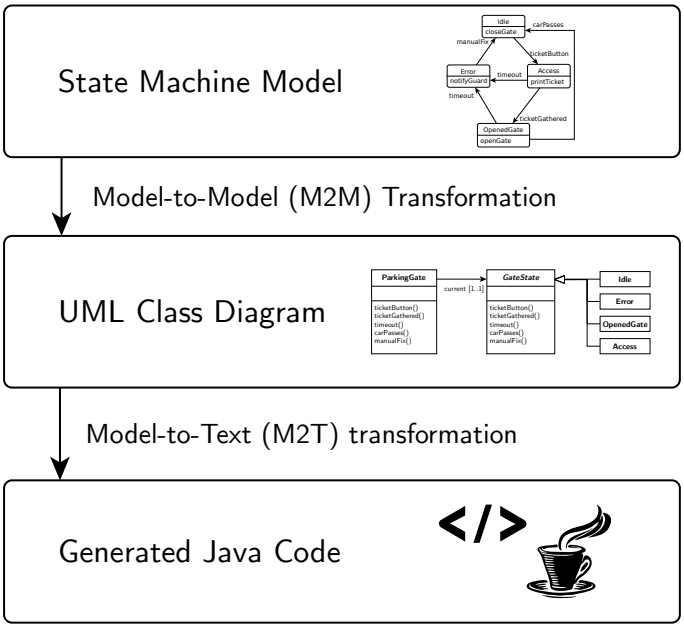


Fig. 1.3 An MDE process to translate a state machine into Java code.

by applying different *model transformations* over the input model. Figure 1.3 shows a two-step transformation process that could be performed to generate the desired code.

In the first transformation of Figure 1.3, a UML class diagram representing the main components of the parking gate state machine is obtained from the state machine model of Figure 1.2. In the transformation, the *State* pattern (Gamma et al. [63]) was applied, and the resulting diagram can be found in Figure 1.4. When following this pattern, a class, which in this case is *ParkingGate*, stores the *current* state of the gate. This state is defined by the *GateState* class. When any of the possible events arrive, i.e., when one of the methods of the *ParkingGate* class is called, this class delegates the call to the current *GateState*, which would be one of the four possible subclasses. Therefore, the logic of the finally invoked method depends on the currently referenced state by the *ParkingGate* object.

Using model transformation techniques, it is possible to define an operation that receives as input a state machine model such as the one of Figure 1.2, and automatically returns a UML class diagram containing the application of the State pattern to the input state machine, such as the diagram of Figure 1.4. As the output of this process is another model, this operation is known as a *Model-to-Model (M2M)* transformation.

From the diagram of Figure 1.4, we can generate Java code that implements the classes contained in the diagram. In this case, we are performing a *code generation* operation from a model, also known as a *Model-to-Text (M2T)* transformation. In this

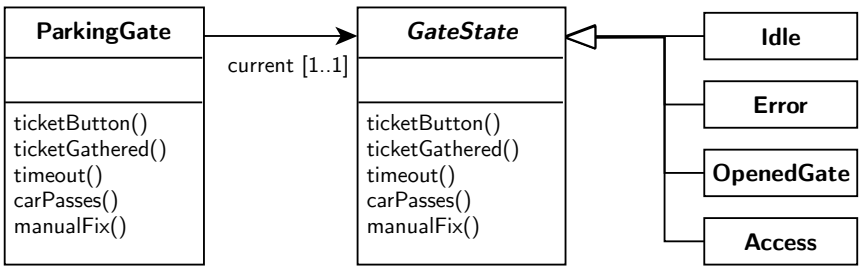


Fig. 1.4 The State pattern applied to the state machine of Figure 1.2.

example, the generated code contains the class definitions and function prototypes for all the methods that each state must implement. This code can be later completed with the final logic of the functions for each state, so this is a semi-automatic transformation. However, when sufficient detail is present in the model, fully-automated code generation operations can also be performed (Monperrus et al. [118])

To make the previous chain of model transformations feasible, models need to be processable by computers. In MDE, models are specified according to a set of rules, which are contained in a *metamodel*. Those models that comply with the rules of a metamodel are said to *conform* to that metamodel.

The issue of defining metamodels to specify a set of rules that models must conform to is similar to the definition of a grammar that programs belonging to a concrete language must follow. Because of this similarity, metamodeling is a popular technique used to define DSLs in the language engineering area ([61, 94, 129]). Next sections describe this area from the metamodeling perspective, which is the one we applied when developing our DSLs.

1.6.2 Domain-Specific Languages Engineering

Usually, several languages are used in combination during the development of a software system. These languages can be organised in two groups, according to their final objective and capabilities.

General-purpose languages (GPLs) can be used to perform a broad variety of operations. The syntax of these languages is not oriented to any particular objective. For instance, Java is a GPL that can participate in the development of very heterogeneous systems, such as web servers or desktop/mobile applications.

On the other hand, *Domain-Specific Languages (DSLs)* are defined to support concrete tasks. The syntax of these languages is tailored to perform these concrete tasks in a specific domain of application. This specificity can provide several benefits.

First, focusing on one domain makes the syntax of DSLs smaller and less verbose than the one of GPLs, as no other tasks have to be supported. In addition, DSLs can offer higher-level syntax constructs, based on concepts and vocabulary from the application domain, that are not reasonable for inclusion in general purpose languages. Lastly, the existence of such constructs brings these languages closer to the jargon of experts in the domain, which makes DSLs a good option for understandability and communication purposes.

There are examples of DSLs in many software-related areas, such as database management (SQL), web development (HTML, CSS) or project building (Maven, Gradle). Moreover, it is also possible to find DSLs in other, non-software domains, such as finance (Christiansen et al. [35]), medicine (Hripcsak [76]), architectural engineering[5], or biology[6].

The capability of DSLs to be adapted to the terminology of a domain has popularised the idea of providing domain experts with DSLs to perform daily tasks. In some cases, these domain experts do not even need to have a programming background. This idea is currently being explored in-depth by the *GEMOC (Globalisation of Modeling Languages)* initiative (Combemale et al. [36]), which has created DSLs for heterogeneous domains such as railroad planing (Vara Larsen and Goknil [161]), farm management or wind power plants monitoring.

With the emergence of MDE, the DSL community also started to adopt a metamodel-based approach to define their languages as a complement to traditional approaches (Fowler [61], Kleppe [94]). Next section describes the process of creating a metamodel-based DSL.

1.6.3 Metamodel-Based DSLs

Two general steps are required to define a new language:

1. Determine what expressions are syntactically correct in that language, this is, specify a *syntax definition*.

2. Provide a meaning for such expressions, i.e., define the *semantics* of the language.

There are different ways to achieve these requirements, depending on the type of DSL defined. During this thesis, we defined DSLs following a metamodeling approach, i.e., models and metamodels are used to formalise some elements of the languages. We

[5] https://technical.buildingsmart.org/
[6] http://sbml.org/

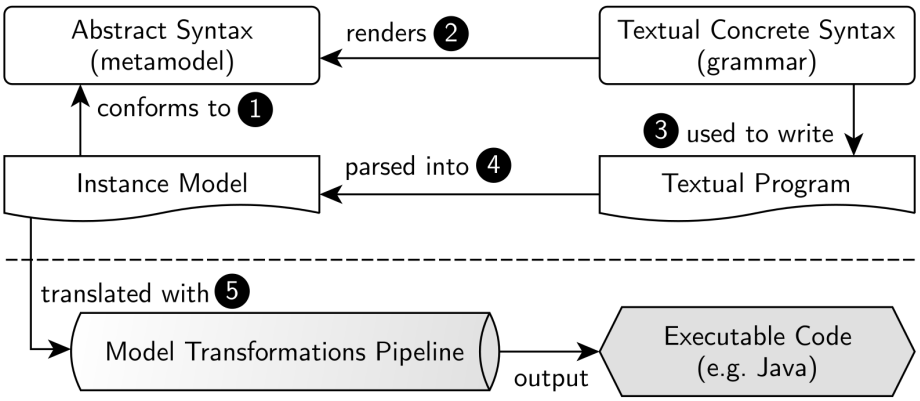


Fig. 1.5 Components of a textual DSL developed following a metamodeling approach and translational semantics.

focused on textual DSLs, so programs are written with these languages according to a grammar. In addition, meaning was given to these programs through *translational semantics* (Kleppe [94]).

The components that conform a language specification with these characteristics appear in Figure 1.5. To illustrate these components, in the following we define a DSL to specify state machines, such as the one shown in Figure 1.7.

Abstract Syntax

The abstract syntax defines the inherent structure of a language. It plays a pivotal role over the different representations, also known as *concrete syntaxes*, that a language might have.

The abstract syntax of an MDE-based language is provided through a *metamodel*. A metamodel specifies the set of rules that models conforming to such metamodel must comply with. These rules define the syntactic structure of a family of models expressed in a concrete modeling notation. Therefore, it could also be said that a metamodel defines the abstract syntax of a family of models or, from another point of view, a *modeling language* (Figure 1.5, relationship 1).

These models and metamodels have to be processable by computers in order to, for instance, take part in automatic model transformations. To allow this processing, the Object Management Group (OMG) defined the Meta-Object Facility (MOF) [125], which is a standard created to define a metamodeling architecture. This architecture is known as the 4-layer architecture, and it is depicted in Figure 1.6.

Elements of each layer conform to some other element from the layer immediately above. For instance, the M0 layer is occupied by those elements or concepts from the

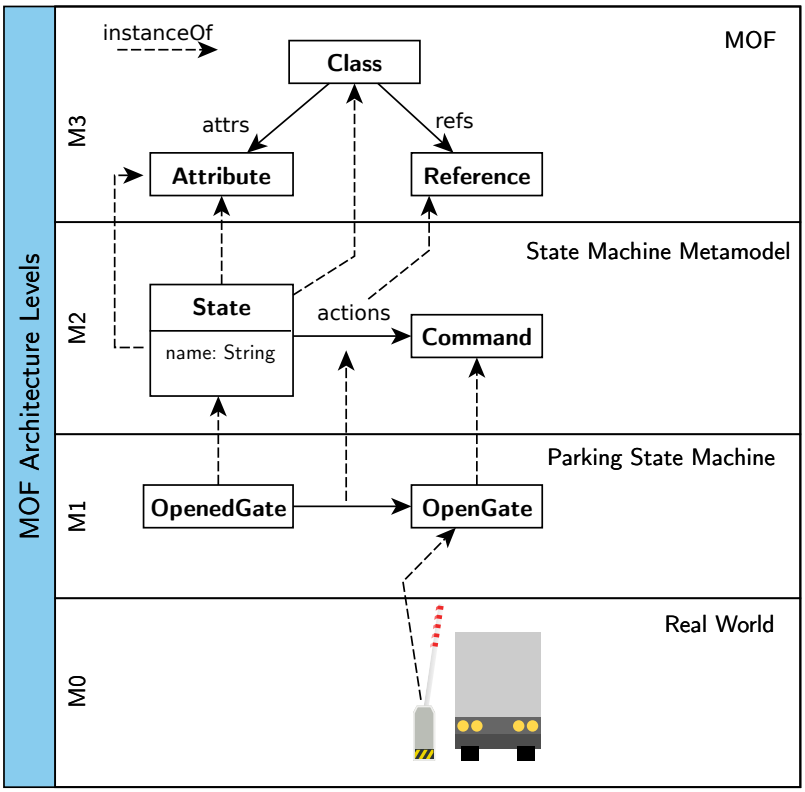


Fig. 1.6 The Meta-Object Facility (MOF) architecture levels.

real world that are being represented by models of the M1 layer. Similarly, these M1 models conform to some metamodel from the M2 layer. The MOF architecture offers a *metametamodel* with the same name, which occupies the M3 layer. A *metametamodel* is a special kind of metamodel used specifically for the definition of other metamodels. For instance, the diagrams that conform the *Unified Modeling Language (UML)* [126] are defined in MOF. Metametamodels conform to themselves, this is, they are defined according to their own syntax. This self-definition allows model management tools to offer a homogeneous treatment of models and their metamodels with independence from their level of abstraction.

Among the available MDE technologies, the *Eclipse Modeling Framework (EMF)* (Steinberg et al. [152]) is considered the de-facto implementation of the MOF architecture. EMF offers the *Ecore* metametamodel, which allows defining metamodels in a class diagram notation that offers a subset of the features defined by MOF. EMF also offers facilities to create and persist models; a reflective API to programmatically manipulate models; code generators to obtain Java implementations from the classes in an Ecore model; and editors for creating and customising model instances.

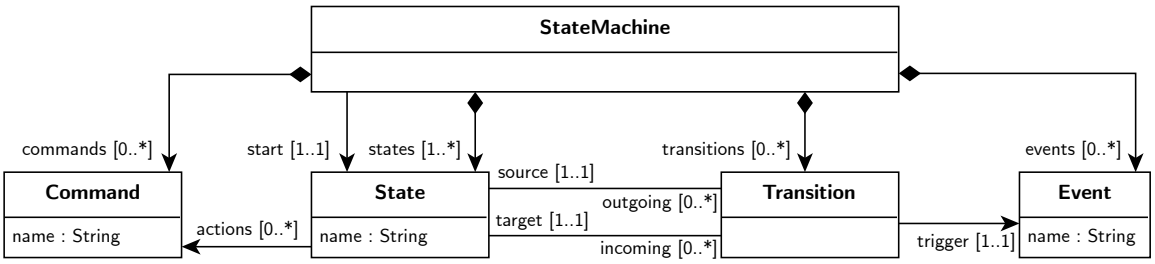


Fig. 1.7 Metamodel to represent state machines.

Figure 1.7 shows an example metamodel expressed in Ecore that allows representing state machines (adapted from Fowler [61]), such as the one introduced in the example of Figure 1.2. According to this metamodel, a *StateMachine* is composed of different *states*, one of which is the *starting* one. Each state has a name, and performs a series of actions or *commands*. Changes between states are managed through *transitions*, which have a *source* and *target* associated states. Each transition is *triggered* by a concrete *event*.

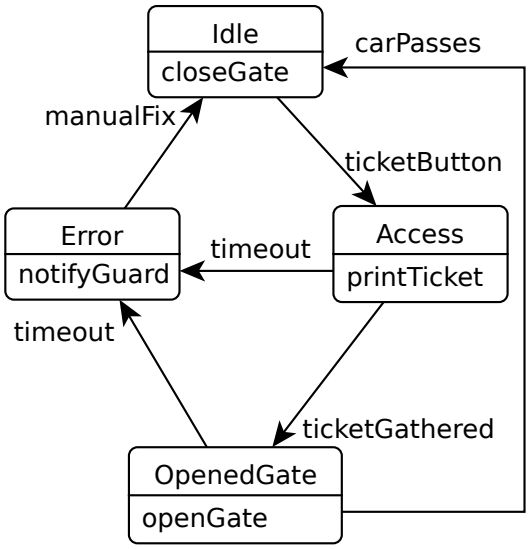
The structural restrictions that a metamodel can enforce may not be enough to guarantee all desired constraints in a concrete context. For instance, in a state machine with a deterministic behaviour, it is not permitted to have two outgoing transitions from the same state that are triggered by the same event. To be able to include these more complex restrictions, constraint languages such as OCL [124] or EVL (Kolovos et al. [100]) can be used to complement the constraints already imposed by a metamodel.

In summary, the metamodel of Figure 1.7 could be considered as the abstract syntax of our language to specify state machines. Next section describes how we can represent and compose state machine models conforming to this abstract syntax.

Concrete Syntax

If we wanted to specify models conforming to the metamodel of Figure 1.7, we would need some sort of textual or graphical representation to author them. Any artefact specified in this representation would be parsed into a model conforming to the target metamodel. These representations are known as *concrete syntaxes*. Concrete syntaxes can be understood as the different renderings or viewpoints of the abstract syntax of a language (Figure 1.5, relationship 2).

Several concrete syntaxes for the same language can coexist. Continuing with the state machine example, Figure 1.8 shows two different concrete syntaxes for the same abstract syntax of Figure 1.7. The example state machine represents the same control




```
state Idle
  actions {closeGate}
  ticketButton => Access
end

state Access
  actions {printTicket}
  ticketGathered => OpenedGate
  timeout => Error
end

state OpenedGate
  actions {openGate}
  carPasses => Idle
  timeout => Error
end

state Error
  actions {notifyGuard}
  manualFix => Idle
end
```

Fig. 1.8 Two concrete syntaxes for state machines: graphical (left) and textual(right).

system of a parking gate that was presented in the previous section. On the left, the graphical concrete syntax that we introduced in Figure 1.2 is shown again, while the concrete syntax displayed on the right is textual. We describe these syntaxes in the following.

As described before, nodes of the graphical syntax represent states, which specify their name and any commands that are performed when entering the state. For instance, the *Idle* state has a *closeGate* command, that invokes any subroutines to perform that action. Arrows in the diagram represent transitions from one state to another. The event triggering the transition is expressed with a label attached to the arrow. So, from *Idle*, we can advance to the *Access* state when the *ticketButton* event is detected, this is, when a client pushes the button at the entrance of the parking to obtain a ticket.

For the textual syntax, the definition of a state starts with the *state* keyword, and ends with an *end*. Commands are expressed by the *action* keyword followed by the command names surrounded by braces. Lastly, transitions are expressed by indicating the event name, and the target state name, e.g., the statement *"ticketButton => Access"* inside the *Idle* state defines a transition from *Idle* to *Access* triggered by the *ticketButton* event.

In this thesis, we have focused on textual DSLs. Programs written with this kind of DSLs must comply with the rules specified by some sort of grammar (Figure 1.5, relationship 3). A grammar can be defined by using a notation such as the *Extended*

⟨*statemachine*⟩ ::= ⟨*state*⟩+
⟨*state*⟩ ::= 'state' name=⟨*STRING*⟩
('actions' '{' ⟨*command*⟩+ '}')?
⟨*transition*⟩*
'end'
⟨*command*⟩ ::= ⟨*STRING*⟩
⟨*transition*⟩ ::= ⟨*event*⟩ '=>' ⟨*stateName*⟩
⟨*event*⟩ ::= ⟨*STRING*⟩
⟨*STRING*⟩ ::= [a-zA-Z0-9]+

Fig. 1.9 Simple grammar that allows defining state machines textually.

Backus Naur Form (EBNF) (Backus [10], Ingerman [79]). Figure 1.9 shows an EBNF grammar for the textual concrete syntax of Figure 1.8 (right).

Grammars are composed of *keywords*, *type rules* and *production rules*. A *keyword* is a reserved word employed in the language expressions. In the grammar of Figure 1.9, keywords are quoted, e.g., *'state'*, *'end'* or *'actions'*. Special characters can also be reserved, such as the braces ('{','}') or the arrow symbol (=>).

Type rules, sometimes denoted terminals, are special rules that are used to define basic types. These rules are usually denoted in upper case, e.g., *STRING*. Types are defined by using a notation similar to the one of regular expressions. For example, the rule *<STRING> ::= [a-zA-Z0-9]+* means that a string is formed by combining one or more characters including lower and upper case letters or digits.

Lastly, *production rules* formalise the syntactic structure that sentences in the language must maintain. To do that, these rules combine keywords, terminal rules and other production rules. Each program written following a grammar always must start from the first rule. In the example of Figure 1.9, the first rule indicates that a *statemachine* is composed by a set of *states*. Each state rule starts with the *state* keyword, followed by a string which is used as identifier. Then, one or more commands and transitions can be specified. To specify them, the *command* and *transition* rules are called, which are also production rules. The special characters *?*, *+* and *** indicate that the preceding rules or text fragments between parenthesis can appear at most once, one or more, or zero or more times, respectively. Lastly, the *end* keyword finishes the *state* production rule.

The definition of a concrete syntax also requires to specify a mapping with respect to the abstract syntax of the language. For textual syntaxes, this mapping is performed by the creation of a parser, which processes programs written according to the grammar

into models conforming to the abstract syntax metamodel (Figure 1.5, relationship 4). As an example, when parsing a program with the grammar of Figure 1.9, the program elements corresponding to the first rule of the grammar would be used to generate a *StateMachine* object. In the same way, each *state* rule would be converted into a *State* instance, and it would be included in the *states* reference of the *StateMachine*. The following string of the rule would be used as *name* of the state, and each *command* and *transition* production rule would be mapped into the corresponding *Command* and *Transition* objects, that would be included in the corresponding references of the *State* object. The rest of the mappings would be performed in a similar fashion.

Language Semantics

The last step to complete a language definition involves creating its semantics, this is, giving a meaning to the programs written in that language. There are different ways for giving semantics to a software language (Kleppe [93]):

- **Denotational**, i.e., by defining mathematical objects or denotations to represent the meaning of a program.
- **Operational**, by describing how a program is interpreted through the sequence of steps that are performed for executing it.
- **Translational**, by converting the programs into other ones conforming to another language, which has well-defined semantics.
- **Pragmatic**, by providing a tool, often denoted as *reference implementation*, that executes any specified program.

From these options, we followed a translational approach: the DSLs we devised in this thesis allow defining high-level specifications of data mining tasks that are translated into low-level code, e.g., Java, through a set of model transformations (Figure 1.5, relationship 5).

Language Workbenches

The components described in the previous section are enough to describe the syntax of a language and its behavior. However, to fully exploit a new language, infrastructure support is fundamental, e.g., a parser to process any defined concrete syntaxes; an editor adapted to the language; autocompletion mechanisms to assist users when specifying language expressions; or an easy way to validate and execute programs. The

costs of developing a DSL along with such an infrastructure are not negligible. Based on these infrastructure needs, the decision to define a new DSL is something that needs to be carefully studied, by weighting the benefits and costs of proceeding with such a definition (Mernik et al. [116]).

To alleviate these development and maintenance costs, several language workbenches are available nowadays, which allow defining the syntax and semantics of a DSL, and then a great part of the infrastructure for the new language is automatically generated. Two workbench solutions were used in this thesis, namely, *ANTLR (ANother Tool for Language Recognition)* (Parr et al. [130]) and *Xtext* (Eysholdt and Behrens [55]).

Given the EBNF grammar of a language, ANTLR automatically provides a parser that allows interpreting languages written following this grammar. Xtext is an ANTLR-based language workbench, and as such allows defining similar, EBNF-like grammars. However, from the definition of one of these grammars, Xtext is able to automatically generate an EMF-based abstract syntax metamodel, and an ANTLR parser that transforms programs written with the provided grammar into instance models conforming to the generated abstract syntax metamodel. If desired, Xtext also allows to manually specify this abstract syntax metamodel. On top of that, Xtext also generates a set of Eclipse plugins that include a full-featured editor based on the Eclipse Rich Client Platform[7]. This editor includes facilities such as code highlighting, validation and autocompletion proposals, and mechanisms to ease code generation and execution. More details of these workbenches will be given when relevant for explaining some aspect of the developed DSLs.

After finishing this description of the MDE-based technologies we used during our work, we present the main contributions originated from this thesis.

1.7 Thesis Contributions

Our work in addressing the research hypothesis formulated in Section 1.5 resulted in the following contributions:

1. We performed a systematic review of the state of the art of the data mining democratisation field [45]. We followed an objective and well-defined procedure (Kitchenham and Charters [91], Wohlin [168]), composed of automatic and manual searches, with the objective of increasing the quality and comprehensiveness of the review. More than 700 works were considered in this review, including both

[7] https://wiki.eclipse.org/Rich_Client_Platform

articles from academia and tools from the industry. This review allowed us to study the strengths and weaknesses of the approaches belonging to this field, and to identify any shortcomings that should be addressed for the purpose of democratising data mining techniques.

The results of this review showed that those easy-to-use analysis solutions that are not adapted to the details of the application domain can exhibit some shortcomings, in particular, a lack of accuracy in the results. Automated approaches, i.e., solutions that try to automatically determine the best configuration for an analysis without requiring any technical input by the final user, are starting to emerge. However, at this moment, these solutions do not offer the same performance as the one a data scientist could achieve. While these automated approaches improve, working in versatile solutions that provide facilities to incorporate any adaptations to the application domain seem to be a good alternative.

2. To alleviate some of the shortcomings detected in the systematic review, we studied the following approach: defining domain-specific languages to invoke data mining processes from a high-level perspective. These languages would hide technical details of data mining process over a high-level, query-based syntax, so that they can be directly employed by non-expert users. Queries generated with these languages would be transformed into analysis processes by using model transformations.

   We developed a prototype language of these characteristics for the educational domain, so teachers could use it to analyse data from their courses (de la Vega et al. [41]). The development of this language revealed some problems of this approach, e.g. the considerable costs that must be confronted to create a language of such characteristics. Nevertheless, we also detected some opportunities, such as the possibility of reusing some components of this language to create analysis languages for other domains.

3. As a result of this previous experience, we defined *FLANDM (Framework to develop LANguages for Data Mining)* (de la Vega et al. [43]), which is an MDE-based framework to create *Data Mining Democratisation Languages (DMDLs)*. This framework exploits the possibility of reusing different components between this kind of languages. For instance, all DMDLs share the same initial abstract syntax metamodel, and the same textual concrete syntax. These generic components are then complemented with meta-information of the application domain, in the form of an instance of an *Entities Metamodel*. This entities model allows end

users to refer to concepts from the terminology of their domain when formulating analysis queries. Additionally, each component of the framework can be easily modified to include any details of interest from a new domain, e.g., to refine how queries are formulated; or to adapt the analysis to the specificities of the domain.

The benefits of FLANDM were evaluated by developing DMDLs for different domains. The results of this evaluation showed that this framework allows reducing the development cost of these analysis languages by 50%.

4. To determine whether the resulting DMDLs could be used by domain experts, we performed empirical experiments with one of the languages generated with the FLANDM framework. Precisely, university teachers coming from heterogeneous areas (computer science, mathematics, and education) used one DMDL to invoke data mining processes over educational data gathered from the e-learning platform that hosted the courses (see Section 1.2.1). The results show that almost all teachers answered the questionnaires correctly after a minimum training.

5. Another issue detected in the review was a lack of works assisting in the data selection and processing steps of an analysis. To mitigate this issue, we developed *Lavoisier* [42]: a high-level language that allows selecting data from a *domain model* that can be easily understood by domain experts. This language automatically processes the selected data into a format that can be digested by conventional data mining tools. For this process, Lavoisier employs a collection of data transformation patterns taken from traditional object-relational mappers; and low-level data transformation operations such as *joins* and *pivots*.

6. Although the Lavoisier language offers an easy-to-use syntax that should be accessible for most domain experts, the degree of control offered by this syntax may not be enough for some advanced operations, i.e., we sacrificed some syntax powerfulness in exchange for better usability. For some users, such as programmers, the existence of advanced constructs might be desirable. For this reason we developed another language, *Pinset* [44], for the purposes of extracting data from domain models and model-based elements. This language includes advanced syntax constructs, such as conditional/loop blocks or first order logic operations, for allowing a more fine-grained control of the extraction process.

1.8 Document Structure

Here we describe the structure followed by the remaining chapters of this dissertation.

Chapter 2 presents our systematic review of the state of the art of the data mining democratisation field (de la Vega et al. [45]).

Chapter 3 presents our FLANDM framework for the development of DSLs for data mining democratisation (de la Vega et al. [43]).

Chapter 4 describes Lavoisier and its transformation patterns for a high-level selection and preparation of data for an analysis (de la Vega et al. [42]).

Chapter 5 describes Pinset and its syntax for advanced extraction of datasets from models (de la Vega et al. [44]).

Chapter 6 describes how we carried out some empirical experiments to assess the validity and utility of our contributions for the data mining democratisation field. The results of these experiments are not yet published, and they will be submitted for revision in the near future.

Lastly, Chapter 7 recapitulates the contributions of this thesis work, and enumerates possible research lines to stimulate new work in the field of data mining democratisation.

Chapter 2

Literature Review

As starting point of this thesis work, we performed a state-of-the-art review of the data mining democratisation field. To ensure the quality of this review, we defined and followed a systematic and objective procedure. This procedure was defined according to the guidelines proposed by Kitchenham [91], which we complemented with the *snowballing* techniques proposed by Wohlin [168, 169], so that comprehensiveness, objectivity and reproducibility of the review can be assessed.

The concrete goals of this review were the following:

1. To assess whether data mining and knowledge discovery techniques are ready to be used by general decision makers.

2. To identify strengths and weaknesses of the different approaches that constitute the current state of the art.

3. To highlight any topics that should be addressed for the purposes of democratising data mining techniques.

This review complements the existing survey of Serban et al. [149], where the types of *Intelligent Discovery Assistants (IDAs)*, these are, solutions to assist analysts in the execution of data mining processes, were described and compared. Our work presents the following benefits over the contributions of this previous survey: (1) it updates the available information on the area, by including the latest 7 years of research (the previous survey was submitted in 2012); (2) we focus on studying the analysis democratisation issue for users without any experience in data mining, whereas most described IDAs of the previous survey are for intermediate and experienced analysts; and (3) our work was performed following and objective and systematic review method,

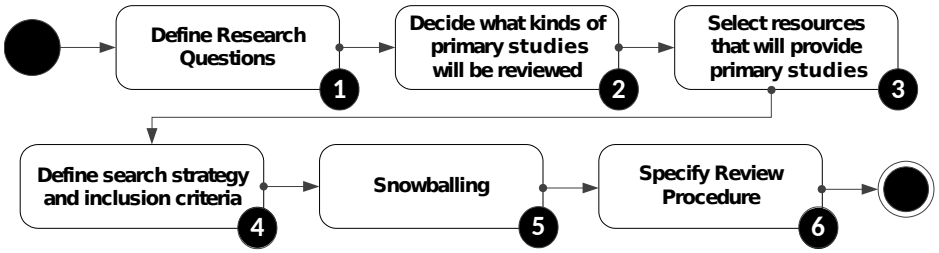


Fig. 2.1 Process for the development of the review protocol.

where automated searches of scientific databases were included to offer a comprehensive view of the data mining democratisation area.

For the purposes of this review, we analysed about 700 data analysis tools and academic articles. This combination of state-of-the-art software and research publications allows us to present a comprehensive view of what is currently being offered in terms of data mining democratisation by the industry, and of what might be available in the upcoming years from the latest works from the academia.

The structure of this chapter is as follows. Section 2.1 describes the review method that we applied to perform this analysis. Then, Section 2.2 comments on the obtained results, and in Section 2.3 we use these results to answer our initial research questions. Finally, in Section 2.4 we conclude this chapter by recapitulating the contributions and discoveries of our review.

2.1 Review method

This section describes the review protocol we employed to study the current state-of-the-art of data mining democratisation. To ensure comprehensibility, objectivity and reproducibility of this work, this protocol was designed according to the guidelines proposed by Kitchenham [91] and Wohlin [169].

The review process of this protocol is described in Figure 2.1, and it can be summarised as follows:

1. First of all, the research questions that should be answered after conducting the review are defined.

2. Then, based on these research questions, the kinds of *primary studies*, e.g., research papers, that will be reviewed are determined.

3. Next, the resources that will be used to find these primary studies, such as digital libraries, are identified.

Table 2.1 Research questions to be answered by this systematic review.

Code	Question
RQ0	What approaches tackle the problem of data mining democratisation?
RQ1	When using the approaches identified in `RQ0`, what actions do decision makers need to carry out to analyse a dataset?
RQ2	What technical knowledge is required to carry out the actions?
RQ3	Can non-expert users make use of data mining tools and techniques by themselves?
RQ4	What trade-offs need to be considered for achieving data mining democratisation?
RQ5	What should be improved in current state-of-the-art so that decision makers can properly analyse datasets by themselves?

4. For each selected resource, a *search strategy* is created. This search strategy must define an *inclusion criteria*, which specifies precisely and objectively the reasons why a primary study should be initially considered for inclusion in the review. Then, each primary study is individually analysed to check whether it adheres to the purpose of this review. If it does not, the study is excluded. The reasons behind these exclusions are specified in an *exclusion criteria*.

5. Finally, to ensure comprehensiveness of our search, the selected primary studies are used as input of a *snowballing* process. This process analyses backward and forward references of the primary studies, to identify new studies that might had not been found using our initial search strategy. These new studies are checked against the defined inclusion and exclusion criteria and, if they fulfil these criteria, they are added to the list of primary studies to be analysed. Then, the included studies are used as input for a new iteration of the snowballing process. If, after an iteration, no suitable primary studies are found, the process stops.

6. Finally, a precise and unbiased evaluation procedure for assessing each selected primary study and answering the research questions is defined.

Next subsections provide more details about how each one of the steps of Figure 2.1 was accomplished.

2.1.1 Step 1: Research Questions

Table 2.1 shows the research questions that this systematic review aims to answer. The ultimate objective of this review is to know how far we are from data mining democratisation (RQ3), and what should be done to reach that goal (RQ5).

To gather evidence for answering these high-level questions, we started by answering first the more fine-grained questions RQ0–RQ2. RQ0 aims to determine the size and maturity of the data mining democratisation community. Assuming that there are approaches that tackle this problem, RQ1 aims to identify the steps that decision makers need to accomplish to analyse a dataset by themselves. Based on this information, RQ2 aims to identify the minimum skills that decision makers need to have to perform an analysis by themselves. The answers to these questions will determine whether inexperienced decision makers can be expected to properly analyse datasets without the help of a data scientist, answering RQ3.

To achieve data mining democratisation, some trade-offs between quality attributes need to be addressed. For instance, data mining algorithms can be made more accessible to non-experts by preconfiguring some of their parameters. On the other hand, this fixed pre-configuration might reduce the accuracy of the algorithms for some concrete analysis (Wolpert [171]). RQ4 explores how each approach deals with these issues.

Finally, using the answers to RQ3 and RQ4, it would be interesting to identify any limitations in the current state-of-the-art that should be addressed to improve the situation of this field. Because of this reason, RQ5 have been added to our research questions.

Next subsection specifies the kind of materials that will be considered as *primary studies* to provide an answer to these questions.

2.1.2 Step 2: Types of Primary Studies

To answer the previous questions, two kinds of primary studies were considered: (1) state-of-the-art-data analysis tools; and (2) research articles on data mining democratisation.

By reviewing state-of-the-art data analysis tools, we expected to get an overview of what a decision maker can currently do with these off-the-shelf software solutions; whereas the review of research literature should provide us a vision of what might analysis tools be able to do in the near future, when existent research results of the academia are transferred to the industry.

2.1.3 Step 3: Search Resources

We used different resources depending on the kind of primary studies that we were looking for. The following describes these resources.

Data Analysis Tools

For finding data analysis tools, typical resources, such as scientific databases, e.g., *Scopus*, were not helpful. This was expected, since tools are rarely reported as scientific articles and, consequently, they are not contained in these databases.

Therefore, we opted for carrying out a survey among several experts in the area, to discover how to perform a systematic and comprehensive search of these tools. Almost all of these experts recommended us to use the *KDnuggets*[1] website. This website maintains highly comprehensive and up-to-date lists with more than 100 data analysis tools and libraries. After checking the completeness of these lists, we decided to use them as the resources for finding the tools that would be reviewed. Precisely, we used the following lists:

1. The main tools list[2], which contains both commercial and free/open-source software applications for data analysis.

2. A list enumerating software that performs *Automated Machine Learning*[3]. Solutions of this kind aim at automatically providing data analysis assets, e.g., prediction models, without the intervention of an expert.

Research Articles

For the discovery of research articles, and according to the guidelines provided by Brereton [26], we defined a preliminary list of scientific databases for performing an automated search. These databases are shown in Table 2.2. Moreover, as recommended by Webster [164] and Jorgersen [82], manual search methods were used to find research works published in conferences, workshops or other venues, as some of these venues might not be indexed by scientific databases. To find these works, a list containing the main conferences on data mining and knowledge discovery was elaborated with the collaboration of external and independent researchers of the area. This list was complemented with some workshops specifically related to the topics of this survey. Table 2.3 shows the list of selected venues for the manual search of primary studies.

[1] https://www.kdnuggets.com
[2] https://www.kdnuggets.com/software/suites.html
[3] https://www.kdnuggets.com/software/automated-data-science.html

Table 2.2 Candidate scientific databases, with their search results.

Database	#Search results
ACM Digital Library	238
IEEE Xplore Digital Library	174
INSPEC	192
Science Direct	1357
Scopus	491
Springer Link	5456
Web of Science	190
Wiley Online Library	324

Table 2.3 Conferences and workshops used as resources in the manual search.

Code	Name
Conf 01	European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML/PKDD)
Conf 02	International Conference on Data Mining (ICDM)
Conf 03	Conference on Information and Knowledge Management (CIKM)
Conf 04	Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD)
Conf 05	SIG Conference on Knowledge Discovery and Data Mining (SIGKDD)
Work 01	Languages for Data Mining and Machine Learning (LML)

Summarising, three kinds of resources were used for finding the elements to be reviewed in this work: (1) the lists from the *KDnuggets* website; (2) a set of scientific databases; and (3) a list of conferences and workshops. Next section describes how our search was carried out using these resources.

2.1.4 Step 4: Search Strategy and Selection Criteria

Next sections describe how each resource was individually processed, according to its own particularities, to find those studies that were later reviewed.

Data Analysis Tools

Each data analysis tool from the lists provided by *KDnuggets* was initially considered as a potential primary study for the review. Thus, the *inclusion criteria* for these tools was simply their appearance in the selected lists, which gave us a total of 138

Table 2.4 Exclusion criteria for data analysis tools.

Code	Description
EC1.1	The tool is deprecated.
EC1.2	The tool requires advanced computing skills to be used.
EC1.3	The tool requires some customer-specific development.

candidates to check. This number corresponds with the last time we audited the lists (June 2018).

The tools were reviewed individually, to discard those that were not helpful for the purpose of this review. Tools were discarded when they exhibited one or more items of the exclusion criteria depicted in Table 2.4.

Deprecated tools that are not longer maintained were discarded (*EC1.1*), since we understood that this deprecation was either because the tool was not useful at all, or because it had been superseded by a similar tool. Therefore, the analysis of this posterior, more successful tool should be enough.

In addition, tools that require advanced computing skills were also discarded (*EC1.2*). For instance, programming libraries for knowledge discovery such as MLC++ (Kohavi et al. [97]) were removed. These tools are designed specifically for developers and programmers, and not for being used by decision makers, so they are out of the scope of this review.

Finally, it was detected that some tools from the *KDnuggets'* lists were not tools exactly, but companies that offer some services. As an example, *ThinkAnalytics* is a company specialized in recommender systems. If a business user wants to acquire its product, they must contact this company, which will customize it for them. However, the product cannot be acquired without the customisations. These customisations would be similar to the process of hiring a data scientist to analyse a dataset on behalf of a decision maker, that is what we try to avoid in this review. Therefore, this kind of tools was also discarded (*EC1.3*).

After this step, 28 tools were finally selected as primary studies for further analysis.

Scientific Databases

According the guidelines provided by Kitchenham [91] and Wohlin [170], a search string for executing an automated search in the scientific databases was constructed. This string is depicted in Table 2.5. To avoid bias and ensure comprehensiveness, this search string was submitted for review and approval to two external data mining experts.

Table 2.5 Search string used in the scientific databases.

Major Terms	Search Terms
Data Mining	(("data mining" OR "knowledge discovery") AND
Usable by non-experts	("democrati*" OR "non-expert*" OR "user oriented" OR "user-oriented" OR "user centered" OR "user-centered")) OR
Business Intelligence	"self-service business intelligence")

The goal of this search string was to retrieve articles where: (1) either the final users were taken into account when developing a data analysis system; or (2) these final users were able to tweak some aspects of the data analysis process by themselves. This search string was iteratively constructed and refined. First, as many related terms as possible were included to make the search highly comprehensive. However, the number of returned results was extremely high, and these results included a lot of work that was not related to the topic of this review. For instance, the inclusion of terms describing easiness of use, such as "friendly", "user-friendly" or "usable", added a cumbersome number of articles which were outside the scope of this review. So, these terms were skipped to increase accuracy of the results.

Although we focused on works belonging to the data mining field, data mining techniques are sometimes employed in the *Business Intelligence (BI)* (Negash [122]) area. BI technologies are used to gain insights from data stored by a company, usually by creating reports that help decision makers visualize and understand some indicators about the performance of a business or process. To achieve this goal, these reports aggregate data from the available sources and perform some descriptive analytics, e.g., statistics of the performance indicators. These reports can also offer facilities to navigate through these data. In some cases, the reports are enhanced with richer information coming from the application of data mining techniques. Recently, several researchers and practitioners have started to work in a new area called *Self-Service Business Intelligence (SSBI)* (Alpar and Schulz [5], Bani-Hani et al. [12], Imhoff and White [78]), which aims to provide decision makers with user-friendly tools to create BI reports by themselves. Based on this recent interest, special terms to gather works from this area were also included in the search string.

We applied the search string in the selected databases to the title, abstract, and keywords of scientific articles. For the sake of comprehensiveness, the search was

Table 2.6 Exclusion criteria for articles in scientific databases.

Code	Description
EC2.1	The work is a position paper.
EC2.2	The work does not address any steps of a data mining process.
EC2.3	The work is not oriented to users outside the data mining area.
EC2.4	The work is not designed to be used with arbitrary datasets.

not limited to any particular discipline, as recommended by Kitchenham [91]. For instance, articles related to the topic of this review might be published in medical journals. Moreover, the search was limited to those articles that, in addition, satisfied the following inclusion criteria: we included peer-reviewed articles, written in English, whose publication date happened up to June 2018.

Table 2.2 shows, besides each considered database, the number of results returned for our search string. As it can be observed, some databases, such as *Science Direct* or *Springer Link*, returned a very large number of results. Nevertheless, most of these were not of interest for our review. For instance, the results included topics such as mutators for genetic programming or latency-based issues of wireless network, which are not connected to the topic of this review. So, we opted for using a subset of these candidate databases, which offered a good balance between accuracy of results and coverage of scientific journals and conference proceedings.

With this premise in mind, *Scopus*, *Web of Science* and *INSPEC* were selected. Before discarding *Science Direct*, *Wiley Online Library*, *ACM Digital Library*, *IEEE Xplore Digital Library*, and *Springer Link*, it was checked that relevant journals and conference proceedings indexed by these databases were also indexed by the ones we selected.

The selected databases returned an initial number of 873 articles. After a cleaning process, where we removed duplicated articles and most invalid results (e.g., table of contents of some conferences showed up as result entries), 559 articles were finally selected as candidate primary studies.

The candidates were individually reviewed to select those that fitted with the purpose of the review. The selection process can be summarised as follows:

1. First, we read the abstract of each article. Those articles that were considered clearly out of scope were discarded. When in doubt, articles were included for further analysis.

2. Then, for the remaining articles, we obtained and read the full versions of each work. Again, those articles that did not fit with the purpose of the review were eliminated.

3. Finally, articles written by the same authors and featuring the same line of research were grouped, and the most mature and comprehensive work of each group was selected.

Table 2.6 specifies the exclusion criteria that was used for discarding research works. First of all, position papers that just state the need for data mining democratisation but that do not describe any approach to achieve it, e.g., Lu et al. [107], Vanwinckelen and Blockeel [160], were left out of the review.

Secondly, we were interested in works about data mining. We did not require the contributions of the selected works to address the whole data mining process, but at least they must address one step of this process, such as data preprocessing, or algorithms selection and execution.

The third criteria for exclusion is determined by the review's focus: we discarded those works that showed clear indicators of being oriented for experts, e.g. articles describing the internals of new analysis algorithms, or presenting utilities that required the knowledge of advanced data mining concepts for their configuration and usage.

Finally, our preliminary searches detected some articles that described software applications for the analysis of data from a concrete domain. These applications were designed to be used for experts in that domain, who had no knowledge in data mining. Therefore, a special effort was made to hide any low-level analysis details to these users. Data mining experts were the ones developing these applications. In the development, these experts addressed exclusively a very specific problem of a concrete domain, without aiming to make the resulting application reusable for other domains or datasets. For this reason, we considered that these approaches, which we will refer to in the following as *ad-hoc applications*, do not fit at all with the purpose of this review, and they cannot be properly analysed using the review procedure that we describe in the next section. Therefore, these approaches were discarded. Nevertheless, there are some concrete contributions of these works that might help achieve data mining democratisation. To make this review more comprehensive, and in case the reader is interested, these contributions are summarised in A.

After the end of the automated search, 15 articles that address the data mining democratisation issue from a broad and generic perspective were accepted as primary

studies for this review. In addition, we detected 11 ad-hoc applications that fall under the description of the previous paragraph.

Conferences and Workshops

Before looking at the proceedings of the conferences and the workshop listed in Table 2.3, we checked whether these proceedings were already indexed in the *Scopus*, *Web of Science* or *INSPEC* databases, to avoid doing redundant work. All of them were already indexed by these databases, but the proceedings of the 2015 edition of ICDM (`Conf 02`) and the workshop proceedings. So, we reviewed these venues manually. Each article was checked against the exclusion criteria contained in Table 2.6. As a result, one article was selected for full review (Vanwinckelen and Blockeel [160]), but it was later excluded because it was oriented for computer and statistics-savvy users.

2.1.5 Step 5: Snowballing

To ensure comprehensiveness of our search process, and following the guidelines proposed by Jalali et al. [80], we used the initially selected 26 primary studies from the automated search as input of a *snowballing process*. During this process, articles cited by and citing the primary studies were analysed. The goal of this process was twofold:

1. To discover any work that should be included in the review, but that had not been retrieved by the automated search.

2. To find follow-up articles of primary studies reporting a more mature work.

The process was achieved by checking one level of backward and forward references of the primary sources. *Scopus* and *Google Scholar* were used to find the forward references, i.e., articles citing the primary studies. For each backward or forward reference, the exclusion criteria of Table 2.6 was applied.

As a result of the first iteration of this process, two new articles (Chittaro et al. [34], Guyet et al. [67]) were added to the list of ad-hoc applications. In addition, an article describing another ad-hoc application (Peng et al. [132]) was superseded by a more recent publication of the same authors [133].

The snowballing process was repeated using the newly found articles as input. As a result of this second iteration, no new work was identified, which implied the stop of the snowballing process, and the end of our article search. Finally, a total of 28 articles was found in our search, of which 15 articles were selected as primary studies for their review, and 13 articles conformed the list of ad-hoc applications (see A).

Table 2.7 Questions to assess stage assistance during a data mining process.

EQ1.1	Is this stage covered by the approach?
EQ1.2	How is the decision maker assisted during this stage?

The selected 15 articles, combined with the 28 tools previously selected from the *KDnuggets* lists, add up to 43 primary studies that were analysed in this work. Next section describes how this analysis was carried out.

2.1.6 Step 6: Evaluation Procedure

A systematic and objective procedure was designed to analyse the selected primary studies. This procedure consisted on gathering a set of indicators, which were grouped into two main categories. These categories are described below.

Assistance During the Data Mining Process

The objective of this first set of indicators was to provide an answer for the research questions `RQ1` and `RQ2`. For each approach, and for each stage of a data mining process (see Section 1.3), questions of Table 2.7 were answered. Namely, we considered the *Data Collection*, *Preprocessing*, *Data Mining* and *Interpretation* stages. *Refinement* of results, this is, the possibility of exploring new issues based on results of a previous data mining process, was also considered as another stage of the data mining process.

Analysis of Trade-Offs

As commented in Section 2.1.1, achieving data mining democratisation implies facing trade-offs between different quality attributes. So, we analysed how a set of these attributes are satisfied by each selected primary study. More specifically, we focused on those quality attributes that typically conflict in the case of data mining democratisation: adoption cost, accuracy of the solutions, functional completeness (i.e. what types of data mining techniques are offered by each approach, and what types are not), and evolution capabilities. These quality attributes, as well as the questions we performed during the data collection, are described in Table 2.8. The results of this analysis should provide an answer to the research question `RQ4`.

Table 2.8 Evaluation questions for the quality attributes analysis.

Quality Attribute	Questions	
Adoption Cost	EQ2.1	Can the approach be deployed in a new domain without requiring some adaptations?
	EQ2.2	Can non-experts carry out any required adaptations without the help of an expert?
Accuracy	EQ2.3	Are the provided results as accurate as possible, i.e., can they be similar to what an expert could achieve manually?
	EQ2.4	Can the analysis be tuned to produce more accurate or precise results?
	EQ2.5	Can non-experts perform that tuning by themselves?
Completeness	EQ2.6	What analysis techniques are available in the approach?
Evolvability	EQ2.7	How easy it is to extend the approach to cover new user needs?
	EQ2.8	Can new analysis techniques be incorporated into this approach?

2.2 Results

This section describes the results gathered after reviewing and evaluating the selected primary studies. For this purpose, we distributed these studies into four categories, which were then analysed. First, we show a summary of these categories, and give some explanation on how primary studies were grouped. Then, each category is evaluated using the procedure described in Section 2.1.6.

2.2.1 Classification of Selected Studies

For the sake of brevity, we do not describe each selected primary study individually. Instead, these studies were grouped according to their similarities, and then they were analysed and compared as groups. We identified four different groups or categories, which are described in the following sections. The correspondence of each primary study with its category is provided in Table 2.9.

Table 2.9 Categories of the approaches that address data mining democratisation.

Category	Primary studies
Workflow-based tools	AdvancedMiner, Alteryx, Angoss Knowledge Studio, BDB Predictive Workbench, Coheris SPAD, Dataiku, Exeura Rialto, IBM SPSS, KNIME, Orange, Partek, Rapidminer, Weka.
Self-Service Business Intelligence	Schuff et al. [147], Behringer et al. [17], Sulaiman et al. [153], Schlesinger and Rahman [146], Abelló et al. [1], Microsoft Power BI, Tableau, IBM Watson Analytics.
Black-Box Components	Campos et al. [28], Reif et al. [137], Ankerst et al. [7], Bilalli et al. [21], Han and Leung [72], Automatic Business Modeler, AutoDiscovery, Auto-Weka, Bicedeep AI, DataRobot, DMWay, Emcien, ForecastThis DSX, Featuretools, Kogentix, MLJAR, Xpanse Analytics.
Development Frameworks	Ben Ayed et al. [20], Espinosa et al. [52], Zorrilla and García-Saiz [176], Alonso and Mencar [4], Santos et al. [145].

Workflow-Based Tools

This category is composed entirely of state-of-the-art tools that are based on the following approach: they provide a set of building blocks, where each block performs an analysis-related task. Users connect these blocks graphically to create what is known as a *workflow*, which specifies a data mining process. The idea is that data mining processes can be specified faster and in a more friendly way by means of dragging and dropping some of these prebuilt blocks, which are later tuned according to the particularities of each concrete analysis. To support this tuning, building blocks have some configurable parameters that can be adjusted.

To illustrate this category, we have selected *Orange*[4] because, in our humble opinion, it is one of the most usable tools and a well-known representative of this category, and it is freely available. Figure 2.2 shows an Orange workflow example specifying a data mining process.

In this workflow, information about clients of a bank is used to predict whether they will subscribe to a term deposit offer before contacting them via phone call. A dataset containing historical information about the clients (e.g. age, current balance, has an mortgage, credit status, whether previous marketing calls were successful) is loaded with a *File* block, which in the workflow of the figure has been renamed to *Clients Data*. The contents of this dataset can be visualised using a *Data Table* block.

[4] http://orange.biolab.si

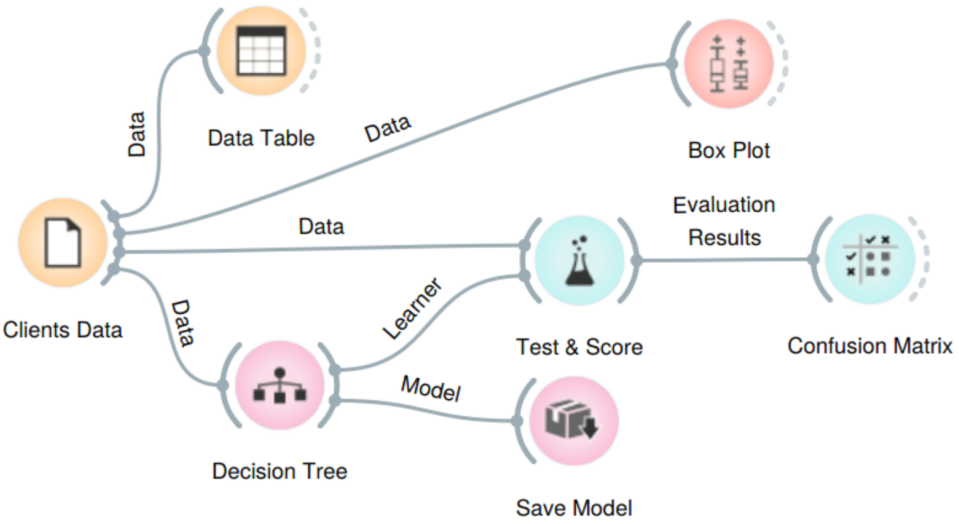


Fig. 2.2 Data mining process example specified with an Orange workflow.

More detailed information about these data can be obtained with the *Box Plot* block that, as it could be expected, draws box plots of each column. Based on the loaded dataset, the *Decision Tree* learner block trains a prediction model, which can be used to estimate the answer of a client to the subscription suggestion. To evaluate the prediction accuracy of this model, the *Test & Score* block performs a cross-validation process (see Chapter 5.3 of Witten et al. [167]) with the provided data. The *Confusion Matrix* shows the comparison of the obtained predictions against the correct ones in a compact and friendly way. When satisfied with the performance of the learner, it can be stored for future usage with the *Save Model* block.

Each block offers some configurable parameters to tune the analysis process. For instance, for the *Decision Tree* block, a maximum depth of the output tree can be specified in order to avoid creating too large and overfitted decision trees. Similarly, for the evaluation block, the number of folds used by the cross-validation process might be changed.

Just from the concepts appearing in the description of the previous example, it can be perceived than a sound knowledge of the techniques employed by these blocks might be required to appropriately select, connect and configure them.

Self-Service Business Intelligence

The term *Self-Service Business Intelligence (SSBI)* (Alpar and Schulz [5], Bani-Hani et al. [12], Imhoff and White [78]) refers to a research and industry field where decision makers that use BI services are provided with user-friendly tools to create reports by

themselves, making unnecessary the intervention of BI experts. Several companies are adapting their BI solutions to include self-service components for non-expert users. Representatives of these solutions, such as *Tableau*, have been studied and included in Table 2.9.

Currently, SSBI tools allow decision makers to perform the following process: First, data is loaded into the tool through the appropriate data source connectors. These connectors allow extracting data from different sources, such as a relational database or an Excel sheet. Then, the imported raw data is filtered and processed. Finally, the information is organized into a report or dashboard where it can be easily visualised and digested.

Continuing with the example of bank marketing calls, using BI, we could load a dataset containing the data of the clients with a CSV (*Comma-Separated Values*) file connector. Then, we might filter the data to focus only on clients that previously accepted a term deposit offer. A report using textual explanations combined with graphs like *scatter plots* or *dice charts* might be created to visualise key performance indicators (KPIs) such as the *Cold Calling Success Ratio*, i.e., the ratio of successful calls to potential clients; aggregated marketing results by country or state; or trend estimations for future seasons. All this process would be performed by non-experts navigating through menus and wizards in a high-level graphical interface. This process is hard to synthesize in a single image, so a picture illustrating it such as Figure 2.2 is not provided for the sake of simplicity.

Researchers are trying to improve the current state-of-the-art of these tools by improve the way in which end users interact with these systems. For instance, some authors are working on making the management of the available data easier (Abelló et al. [1], Schlesinger and Rahman [146], Sulaiman et al. [153]); and on increasing the variety of analysis that can be performed (Behringer et al. [17], Schuff et al. [147]).

In summary, this group is somehow similar to the *Workflow-Based Tools*, but it is more oriented to the creation of dashboards and reports, instead of particular data mining processes.

Black-Box Components

This category groups research work and tools whose goal is to hide details of data mining processes, so that they can be carried out by non-expert users. We named these approaches as *black-box components*, because (1) they can be applied as are to new contexts, without any parameter tuning or adjustments (i.e. they offer black-box

functionality); and (2) they offer support of some stages of the analysis, but not for the process as a whole, so they can participate as components of an analysis.

Most research work in this category focus on facilitating the mining stage (Ankerst et al. [7], Campos et al. [28], Han and Leung [72], Reif et al. [137]), but there are also some approaches that deal with the data preprocessing stage (Bilalli et al. [21]).

A clear representative of this category is the work of Campos et al. [28], where a data mining module extension for the Oracle Database Management System is presented. This extension offers some procedures that execute prebuilt data mining processes using as input the data contained in a Oracle relational database table.

For instance, the *PROFILE* procedure allows obtaining similarities of those records which share the same value for a specific column. Continuing with the bank marketing example, the *PROFILE(clients_information, call_outcome, results)* command might be executed to analyse the *clients_information* table, with the objective of checking what features are shared between those clients who have the same value in the *call_outcome* column. The results of this procedure, which are provided in the output variable *results*, are two sets of rules. The first set describes those clients who are likely to accept the term deposit and the second one those ones who are not.

The main contribution of this approach is that the user does not need to know the details behind these procedures. The user knows neither what specific algorithm is being used to execute the procedure, nor how the algorithm parameters have been exactly configured. Nevertheless, data must be retrieved from its sources, cleaned, and processed to fit into an adequate format, which in this case is a single relational table, before these procedures can be executed.

Regarding other approaches in this group, Reif et al. [137] offers a building block which can be used in a RapidMiner workflow to automatize the selection of a classifier. Ankerst et al. [7] includes computer-based visualization techniques that aid the end user in the creation of a decision tree. Han and Leung [72] presents a web service to automatically find frequent sets of items that often appears together. Finally, Bilalli et al. [21] present a black-box component that, based on the characteristics of a given dataset, applies a set of preprocessing transformations to prepare the data and improve the prediction results of the generated models. As with the previous example, the main contribution of these approaches is that some tasks of a data mining process are automated and carried out by a computer transparently.

Selected tools belonging to this category also try to automate the mining stage of the process, focusing almost completely on prediction analysis. Users of these tools have to provide a dataset with training records, and specify what they want to predict.

Then, the tools automatically build a prediction model without the intervention of any expert. This prediction model can then be used over new records to perform predictions. Examples of these tools are *DataRobot*, *Kogentix* or *MLJAR*. As an exception, one of the tools, *Featuretools* [88], focuses on the preprocessing stage by automating *feature engineering*, i.e. the generation and selection of features that will be used to train a model.

Development Frameworks

This category contains research works that provide methodologies to develop knowledge discovery systems, with the objective of making these systems usable by non-experts (Alonso and Mencar [4], Ben Ayed et al. [20], Espinosa et al. [52], Santos et al. [145], Zorrilla and García-Saiz [176]).

A representative of this category is the work of Zorrilla and García-Saiz [176], who propose to use a service-oriented methodology to develop data analysis systems. The first step in this methodology is to know which questions end users want to answer. Then, the data mining processes that will compute these answers are developed by data scientists and wrapped as web services. These wrapped processes are designed to be as automated as possible, preventing the parameters of their algorithms from needing adjustments due to slight changes in the input data. Finally, user-friendly, web-based interfaces are developed. These interfaces must allow non-experts to invoke the analysis services and receive the obtained results, without requiring any technical knowledge of the wrapped processes.

Using this approach in our marketing example, a preconfigured prediction process to analyse clients data would be initially developed by experts, and encapsulated in a web service. Then, a user-friendly web interface to invoke this service would be developed and deployed. Using this interface, bank employees could determine the likelihood of a client accepting their marketing offer. Since the wrapped prediction process is autoconfigurable, it should also work with datasets coming from other banks, although data in these datasets can exhibit other internal properties, such as a higher number of outliers.

As for the other approaches of this category, Ben Ayed et al. [20] present a development process that combines the Unified Process from Software Engineering (Kruchten [102]) with the U model for Human-Computer Interaction (Lepreux et al. [104]) to generate an iterative, user-centred development method, which outputs knowledge discovery applications that can be used by non-experts. Espinosa et al. [52] propose a methodology that guide users in the development of data mining applications. This

methodology is based on two main elements: (1) a taxonomy of questions that helps non-expert users to identify the data mining technique they should use, i.e., clustering to group data; and (2) a recommender system that returns the best data mining algorithm for a given technique inside a specific context (e.g. Kmeans or Expectation Maximization as clustering algorithms).

Santos et al. [145] present a reference architecture to build data mining systems. This architecture includes: (1) a data warehouse that contains the data to be analysed; (2) an ontology that models the domain to which these data belongs to; and (3) a set of metadata that specifies how the ontology connects to the data warehouse and how to process each data bundle. These metadata are specified with the help of a data mining expert. Once these elements have been created, end users interact with the domain ontology to select those data they want to analyse. Based on the metadata, the system proposes several kinds of analysis, and the end user selects one of them. Then, using the metadata again, the system automatically instantiates a data mining process, configures it appropriately and executes it, returning the results to the end user. Finally, Alonso et al. [4] introduce a development process, based on fuzzy logic, to create data analysis systems whose results can be easily interpreted by end users, providing explanations in natural language.

It should be noted that, although the systems generated using these approaches are initially prepared by data mining experts, once they are ready for use, these experts are not required any more. It could be argued that these systems are similar to the *ad-hoc applications* commented in Section 2.1.4 and A, i.e., applications developed by experts to solve a specific problem in a particular setting; and, therefore, they should be discarded according to our *exclusion criteria* (see Table 2.6). Nevertheless, oppositely to *ad-hoc applications*, these approaches are designed to be applicable to different domains. In addition, the generated applications could also be reused without further adjustments with different datasets from the same domain, whereas *ad-hoc applications* are designed to work with a single and very specific dataset, without reuse across a domain (or in any other different domain) in mind.

2.2.2 Evaluation Results

Now we comment on the results obtained after executing the evaluation procedure described in Section 2.1.6 to the selected primary studies. This evaluation procedure was divided into two stages, being each stage described in its corresponding subsection.

Table 2.10 Coverage of the data mining process stages offered by each category.

Category\Stage Action	Collect	Preprocess	Mine	Interpret	Refine
Workflows	✓	✓	✓	✓	✓
SSBI	✓	✓	(✓)	✓	✓
Black-Box Components		✓	✓		
Development Frameworks	✓	✓	✓	✓	(✓)

Assistance During the Data Mining Process

This first stage of the evaluation procedure (see Section 2.1.6 and Table 2.7) aims to know what stages of the data mining process are covered by each approach, and how well each stage is covered. Results of this analysis are summarised in Tables 2.10 and 2.11. Table 2.10 shows the coverage of the stages of a data mining process, whereas Table 2.11 describes briefly how these stages are covered. In Table 2.10, a stage is marked if at least one of the approaches of the corresponding category addresses it. A check mark surrounded with parentheses indicates that limited support for that stage is present, but it is not as clear as in the other cases. We comment on these results in the following.

EQ1.1: Covered Data Mining Process Stages Workflow-based tools and SSBI solutions cover the whole data mining process, since they offer building blocks or wizards for all the stages. Figure 2.2 provides an example of a workflow where all data mining stages are covered. Moreover, these blocks and wizards typically contain configurable parameters, so the process can be tuned and refined.

Black-box components assist either by automatically selecting data mining algorithms, or by providing users with the ability to execute analysis tasks without knowing their low-level details. In these approaches, only the mining stage is typically covered at the time of writing this survey. Therefore, the user is still in charge of acquiring and preparing the data, interpreting the results of the analysis, and adjusting the input data if some refinements are desired. Exceptions are the work of Bilalli et al. [21] and Featuretools [88], which try to automate the preprocessing stage.

The development frameworks require the intervention of a data mining expert to create some initial elements. Nevertheless, once these elements are ready, the user is assisted for all the data mining stages of the configured processes. However, these approaches does not offer many ways to refine these processes, e.g., the possibility of adding or removing elements from the input data to be analysed is not supported.

Table 2.11 Assistance offered by each category during the analysis process.

Category	EQ1.2: Decision makers are assisted through ...
Workflows	building blocks that allow to graphically configure a complete analysis workflow.
SSBI	a user-friendly interface where high-level menus and wizards allow performing data-related tasks.
Black-box Components	tools that perform a concrete mining stage automatically.
Development Methodologies	rules and frameworks to instantiate systems that can be used by non-expert users.

As an example, Zorrilla et al. [176] developed, following the service-oriented methodology they propose, a high-level system for analysing student performance. Using this system, teachers can, among other things, try to determine reasons why students fail, taking into account any available data of these students, e.g. activity in the e-learning platform, demographic information, or results in partial tests. Nevertheless, the developed analysis system does not allow teachers to decide which data is used to analyse the students' performance, i.e., teachers cannot focus only on activity data to discover performance patterns related with how students use the platform. If we wanted to incorporate this analysis, a data scientist would be again required to modify the tool. Therefore, refinement support is somehow limited in this category.

EQ1.2: How is the decision maker assisted during each stage? We consider now how each category assists end users for the data mining stages it covers.

Workflow-based applications provide building blocks that allow specifying data mining processes graphically. Nevertheless, users are still responsible for selecting the right blocks for each analysis, and from their appropriate configuration and interconnection. To accomplish these tasks, sound knowledge on data mining concepts is required. Therefore, it can be concluded that workflow-based applications do not provide much assistance for the non-expert users we are focusing on this work.

SSBI tools offer some wizards and interfaces to define data analysis tasks. These tasks are mostly oriented to data reporting and descriptive analysis, although some tools have incorporated support for new data-related tasks over the last years, such as the *Prediction Workbooks* offered by Watson Analytics[5]. As it happened with the workflow applications, users are still responsible for executing some low-level tasks, for

[5] http://bit.ly/predworkbooks

which specialised knowledge might be required. For instance, in the general case, users might need to aggregate or normalize data before they can be used as input for a data analysis task.

Black-box components provide simple commands and interfaces that hide all low-level details of certain analysis tasks to the end user. Therefore, these elements can be perfectly executed by users with no expertise in data mining techniques. Nevertheless, only the data mining or data preprocessing stages, as already commented, are currently covered by these approaches.

Lastly, *development frameworks* do not provide any support for executing data mining processes without the intervention of an expert. These methodologies provide some development rules, often associated to a prebuilt infrastructure, to help create data analysis systems ensuring that the resulting products can be employed by end users without expertise in data mining tasks. Therefore, a data mining expert is initially required to follow these rules and to configure appropriately the associated infrastructure, when it is provided. Then, once the system is ready, end users can select data, execute different analysis tasks by themselves, and obtain results that can be easily interpreted. Although this initial intervention of a data mining expert is always required, some of these approaches (Espinosa et al. [52], Santos et al. [145]) aim to reduce this intervention as much as possible.

On the other hand, this initial intervention required in the development frameworks' approaches allows for the obtention of a system with higher accuracy than the achieved, for instance, when using black-box components, because this system has been adjusted to the particularities of the specific domain it is being deployed to. This means that, assuming an extra cost, system accuracy can be increased. This and other trade-offs are commented in the next section.

Analysis of Trade-Offs

This second stage of the evaluation procedure (see Section 2.1.6 and Table 2.8) aims to know how each one of the categories deals with the different trade-offs that are inherent to data mining processes. Table 2.12 summarises the results of this analysis. A dash ("-") appears as answer for those questions where an answer to a previous question invalidates them, e.g., EQ2.1 and EQ2.2.

Workflow-based tools provide building blocks that are designed to work with any input dataset, so they are can be used in any domain without prior adaptation. However, as commented in the previous section, these building blocks are not designed to be

Table 2.12 Results of the quality attributes by category.

Quality Attribute	Workflows	SSBI	Black-Box Components	Development Frameworks
Adoption Cost				
EQ2.1. Can it be deployed without adaptations?	Yes	Yes	Yes	No
EQ2.2. Can non-experts perform the adaptations?	-	-	-	No
Accuracy				
EQ2.3. Can the analysis reach expert-level accuracy?	Yes	No	No	Yes
EQ2.4. Can the approach be tuned to improve accuracy?	Yes	Yes	No	No
EQ2.5. Can non-experts perform the tunings?	No	No	-	-
Completeness				
EQ2.6. What analysis techniques are available?	All	Descriptive	Predictive	All
Evolvability				
EQ2.7. How easy it is to extend the approach?	Easy	Med.	Hard	Med.
EQ2.8. Can new analysis techniques be included?	Yes	Yes	No	Yes

employed by non-expert users, so a data scientist is always required to design the workflows that will implement a specific analysis process.

Building blocks come with a default configuration, so that they can be used with a minimum effort. Nevertheless, this default configuration might not perform well for all domains (Wolpert [171]), provoking that the results of these defaults might not be as accurate as possible. Defaults can be modified to increase the accuracy of the results, but this task requires a deep knowledge of data mining techniques, making necessary again the help of a data scientist.

Regarding evolution, new business questions might be addressed by means of creating new workflows, for which a data scientist would be required. Similarly, existing workflows might be adapted to new requirements by changing the configuration and interconnection of their building blocks. For instance, if after performing an analysis over a dataset, we wanted to repeat that analysis but just for a subset of the input data, this change might be addressed by adding a filter block after the data loading step. Also, new data mining techniques could be incorporated to these tools using extension mechanisms that support the addition of new building blocks, which wrap custom data mining tasks. As before, the intervention of an expert would be required perform any of these tasks.

SSBI tools are available as generic solutions that, in most cases, can be deployed in any context as are and used by decision makers without advanced expertise in data analysis. Using these tools, these decision makers would define the required analysis process by themselves. Therefore, these tools can be classified as domain-independent solutions that can be set up with a relative low effort, and without the intervention of a data scientist. Nevertheless, most SSBI tools, at the time of writing this work, focus mainly on offering reporting capabilities, including basic descriptive analytics processes that can be incorporated to these reports.

The accuracy of SSBI tools might be compromised by the powerfulness of the offered functionality. These tools try to be understandable by a lot of different users with heterogeneous expertise, so some advanced functionalities might be not present to favour the simplicity and amenability of the tool. As an example, for a data preprocessing stage, users of a SSBI tool can only choose between the available list of cleaning tasks, which may not be as large as the plethora of libraries and specialised tools than an expert might employ for this purpose. While, to some extent, this issue might also apply to workflow applications, in our review we have observed that workflow apps are way more complete in terms of offered functionality than SSBI solutions.

In terms of evolvability, the analysis processes initially designed by decision makers can also be modified and refined by themselves. In addition, if new data mining techniques were required to answer new business questions, these techniques might be incorporated into some of these tools through custom scripts. For instance, if we wanted to incorporate support for calculating association rules in one of these tools, we should write a script that implements or invokes the corresponding algorithm. Unfortunately, this kind of task demands the intervention of data mining experts.

Black-box components are devised to work as are for any input, and with no prior adaptation work required. However, as commented in the previous section, these approaches do no cover all stages of a data mining process. For instance, most approaches focus exclusively on the mining stage, and specifically in offering prediction analysis tools. This implies that, for instance, end users are in charge of selecting, cleaning and formatting the data of interest for an analysis into the format accepted by the employed black-box prediction tool. These tasks will be often too far away from the capabilities of decision makers. Therefore, although these approaches can be easily included in new contexts to cover some stages of the analysis process, the intervention of data scientists might be required in order to perform the remaining stages.

This black-box approaches do not take into account any specificities of the application domain during the analysis, which might decrease the accuracy of the obtained results. This lack of adaptations to each domain might return worse results than the ones an expert may achieve. In addition, the black-box nature of these tools might make unfeasible to adapt these applications to a specific domain, even with expert intervention. For instance, most of the tools provide a way to create prediction models, but we might not be able to slightly modify how the prediction model is built, e.g., by deciding which features are more relevant for the prediction. Consequently, these applications cannot be extended to support new user needs, and new analysis techniques cannot be easily incorporated.

Development frameworks are designed to be used in any context, so they can be classified as domain-independent initially. Nevertheless, these approaches often provide an infrastructure that needs to be modified to fit in with the particularities of a specific domain. For instance, in Santos et al. [145], some metadata needs to be specified by a data mining expert to indicate how the different elements of a domain ontology should be processed. Therefore, these approaches need some previous work before being deployed in a new context, and this work must be carried out by an expert.

The counterpart of the higher cost of these approaches is that the initial intervention of experts allows taking into account any relevant details of the application domain,

which contributes to increase the degree of accuracy of these solutions to something very similar to what an expert might achieve. For instance, in the educational example of Zorrilla and García-Saiz [176], although teachers use wrapped processes that are not configurable, these processes have been adapted to the educational domain, which may improve the quality of the results when compared with, for instance, commands of black-box components.

These tools can also be extended to support new business needs. For instance, in Santos et al. [145], the domain ontology might be used to select different subsets of data as we are gaining insights in a domain, and we want to find answers to more precise questions. Nevertheless, not all these approaches support this kind of refinement, and the intervention of experts might be required depending on what new user needs we want to address. Similarly, new data mining techniques can be incorporated to these methodologies and their associated infrastructure with the help of an expert.

With the previous paragraph we finished the description of the obtained results in the evaluation. Next section answers the research questions that we formulated as the objectives of this work.

2.3 Discussion

During the evaluation procedure, we gathered enough evidence to provide answers for our research questions (see Section 2.1.1). These answers can be found below.

2.3.1 RQ0. What approaches tackle the problem of data mining democratisation?

We identified four different categories of approaches that collaborate in the data mining democratisation field: *workflow applications*, *Self-Service Business Intelligence solutions*, *black-box components* and *development frameworks*. More details of these categories and the approaches belonging to them can be found in Section 2.2.1. Although there seems to be an interest in this field, the amount of articles found in the academia is still small, as not many approaches address this subject yet. On the other hand, important enterprises, such as IBM or Microsoft, are starting to perform a non-negligible effort to provide decision makers with user-friendly solutions for performing data exploration and analysis. This means that data mining democratisation is being considered an important issue, which will need to be addressed more in-depth in the near future.

2.3.2 RQ1. When using the approaches identified in the previous question, what actions do decision makers need to carry out to analyse a dataset?

These actions vary highly depending on the kind of approach selected. In workflow-based applications, users connect and configure pre-built blocks to specify a data mining process. In SSBI solutions, users navigate through a set of high-level user interfaces and wizards to configure an analysis process, more based on visualization and reporting than in data mining. Black-box components do not provide support for all stages of the analysis process, so the user needs to carry some of them without any assistance. For instance, user often needs to retrieve, format and clean the data to analyse. For those stages that are automated, users just need to invoke some commands that hide the low-levels details of their execution. In development frameworks, an initial minimum intervention of a data-scientist is required. After that, users can operate the systems by themselves through interfaces abstracted from low-level details.

2.3.3 RQ2. What technical knowledge is required to carry out the actions?

Workflow-based applications require a sound knowledge of data mining techniques, since the offered building blocks are more oriented to data scientists rather than for non-expert users. On the other hand, SSBI solutions are mostly designed to be used by non-experts, so no technical knowledge is initially required. Nevertheless, if advanced analysis are wanted, i.e., going beyond basic reporting, some technical knowledge might be necessary. Black-box components do not require any technical knowledge for the stages of the data mining process they address. Development frameworks require an initial configuration of a certain infrastructure. This initial configuration has to be carried out by an expert but, once it is completed, the resulting system can be operated by non-expert users.

2.3.4 RQ3. Can non-expert users make use of data mining tools and techniques by themselves?

In the light of the answers to previous questions, the answer is no. Workflow-based applications are not designed to be employed by non-experts. SSBI solutions offer a limited support for advanced data mining techniques. When these techniques are addressed, SSBI solutions often face the same problems as workflow-based applications,

i.e., they do not provide suitable solutions for non-expert users. Black-box components can be used by non-experts, but they do not address the whole data mining process. Finally, development frameworks require the initial intervention of a data scientist, although they aim to reduce this intervention to the minimum.

2.3.5 RQ4. What trade-offs need to be considered for achieving data mining democratisation?

Genericity versus accuracy and *accuracy versus cost* seem to be prominent trade-offs for the studied approaches. Those approaches that are absolutely domain-independent, such as the black-box components, do not require the intervention of experts, so they can be adopted with a relatively low-cost. Nevertheless, since they cannot be optimised for taking into account the particularities of any domain, their results might be not as accurate as possible. In some cases, accuracy might be seriously compromised and the system would not fulfil end-user needs. On the other hand, the need of experts to include these domain-specific customisations implies an increase in adoption cost.

2.3.6 RQ5. What should be improved in current state-of-the-art so that decision makers can properly analyse datasets by themselves?

Black-box components seem to be the more promising attempt to data mining democratisation, since the intervention of experts, for those stages these approaches address, is not required. Nevertheless, as previously commented, accuracy is sometimes compromised. Therefore, it would be desirable to find techniques that help to automatically select and configure the algorithms that best fit in with the particularities of each domain. Some works, such as Reif et al. [137] or Billali et al. [21], offer some initial results in this direction. These works belong to the area of *metalearning* (Brazdil et al. [25]), where meta-prediction models are built to help select the best algorithms for an analysis. Other possible solutions are *autoconfigurable* or *parameter-less* techniques (Feurer et al. [58], Zorrilla et al. [177]), which try to tune their parameters automatically to offer the best possible results. Nevertheless, more research work is needed in this area to allow *black-box components* to be completely automated while avoiding any noticeable accuracy loss, facilitating data mining democratisation.

Metalearning and autoconfigurable algorithms have provided interesting solutions for some specific problems and areas, such as classification and regression problems, but

there is still a lack of generic solutions that can be universally applied. While we wait for these global solutions, the possibility of tuning, with minimum expert intervention, an initially generic application, such as development frameworks do, seems to be an interesting and pragmatic solution to the *accuracy versus cost* trade-off. Therefore, the design of generic analysis frameworks that can be instantiated in a concrete domain, with a minimum expert intervention, should be studied more in-depth.

Most of the analysed work focus on data mining algorithm selection and execution, e.g. Ankerst et al. [7], Reif et al. [137]. Only two primary studies, belonging to the black-box components category, dealt with issues on the preprocessing stage (Bilalli et al. [21], Kanter and Veeramachaneni [88]). This shows a lack of work addressing the *data obtention* and *preprocessing* stages of the data mining process, even when these stages have been demonstrated critical for the outcome of an analysis (Crone et al. [38], Munson [119]). Therefore, more research work would be needed in these stages.

2.4 Chapter Summary

This chapter has presented a systematic review that aims to identify how far we are from an effective democratisation of data mining. To achieve this goal, we followed the review protocol proposed by Kitchenham and Carters [91], which we complemented with the *snowballing* technique as proposed by Jalali and Wohlin [80]. In this review, we considered as primary studies both research work and state-of-the-art tools. This combined analysis gave us the complete picture of the field, including what is available now and what new techniques should be expected to be adopted in the upcoming years. During the review, 559 research papers and 138 tools were initially considered, from which we selected 15 articles and 28 tools as primary studies, adding up to 43 (see Table 2.9).

In terms of quantity, it seems that there is a considerable interest in the industry in offering solutions for data mining democratisation. With respect to the academia, although there are some publications in this field, most of them refer to solutions for very specific problems, and as such they are difficult to generalize. The number of solutions found that actually try to improve the situation of data mining democratisation from a general perspective was not that large.

The selected primary studies were grouped in four different categories: (1) *Workflow-based applications*; (2) *Self-Service Business Intelligence (SSBI) solutions*; (3) *Black-box components*; and (4) *Development frameworks*. Each category was then analysed against a well-defined systematic evaluation protocol.

The evaluation concluded that *workflow-based applications* are not designed to be employed by non-expert users, whereas *SSBI solutions* offer limited support for advanced data mining techniques at the time of writing this review. *Black-box components* are perfectly usable by non-experts, but they only address some stages of the data mining process, and might exhibit accuracy flaws. *Development frameworks* try to solve this accuracy problem by means a controlled and reduced intervention of a data mining expert, who would perform domain-specific customisations for a particular setting.

To sum up, it can be stated that, although there are some promising initial steps, we are still far from data mining democratisation. During this thesis work, we developed approaches that try to alleviate the following issues identified during this review:

1. Generic solutions, i.e., those that are completely domain-independent, might exhibit accuracy problems, since they do not take into account the particularities of each domain to configure their algorithms or to preprocess input data.

2. The issue of facilitating the data selection and data formatting stages is scarcely addressed in the literature.

Next chapters describe these approaches.

Chapter 3

FLANDM: A Framework to Develop DSLs for Data Mining

The availability of user-friendly interfaces is one of the challenges we identified for the accomplishment of data mining democratisation. One key component of these interfaces should be a seamless integration with the characteristics of a domain, such as vocabulary or semantics. This way, the developed systems would feel as familiar as possible to the end users.

Focusing on this challenge, we explored the idea of providing non-expert users with DSLs to perform data mining analyses. These languages offer a query-based syntax to specify a data mining analysis process over domain data. This syntax is abstracted of any technical details of the executed data mining tasks, so that the queries that end users have to formulate only contain high-level commands and terminology coming from the application domain. We denoted these languages as *DMDLs*, from *Data Mining Democratisation Languages*.

Although DMDLs seem to be a good solution for hiding analysis complexity, determining whether it is worthy to develop a new DMDL for a given domain is not trivial (Mernik et al. [116]). The definition and maintenance of such a language involves a considerable effort which, although beneficial, might not be affordable for some contexts. We experienced this cost issue when developing our first DMDL prototype, which was devised for teachers to analyse information about their courses.

To alleviate the cost barrier of adopting a DMDL, we present *FLANDM (Framework to develop LANguages for Data Mining)*. This framework allows defining DMDLs for concrete domains with reduced development effort. FLANDM offers a base DMDL infrastructure, which is customized for each deployment. This infrastructure can be

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

Fig. 3.1 Examples of queries written with the educational DMDL.

also used to create prototype languages easily, so that the suitability of DMDLs can be tested in new domains without the need of a hard commitment.

The rest of the chapter is structured as follows. Section 3.1 describes our mentioned first prototype of a DMDL, from which important insights were gathered and applied in the design of FLANDM. Section 3.2 gives an overview of FLANDM, presenting its main components. Sections 3.3 and 3.4 describe the two main parts of the FLANDM framework, namely, the queries specification and queries execution components. In Section 3.5, we perform an evaluation of FLANDM to assess its cost reduction in development and maintenance through the development of several DMDLs. Finally, Section 3.6 summarizes the contents and contributions of this chapter.

3.1 Experience from an Educational DSL

Our first data mining democratisation language (DMDL) was a prototype applied to the educational domain. In this domain, teachers want to get insights of their courses by analysing the information stored in the e-learning platform, such as Moodle (Rice [138]), that hosts each course. These insights could then be used to, for instance, 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. 3.1 shows some examples of queries written in this DMDL. The first query (Fig. 3.1, Q1) invokes the profile extraction process by using the `show_profile` primitive, 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. 3.1, 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. 3.1, 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 (Hall et al. [70]) 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 (Combemale et al. [37], Kleppe [94], Völter et al. [163]). First, its abstract syntax was specified with an *Ecore* metamodel (Steinberg et al. [152]). Then, a concrete textual syntax was defined using *Xtext* (Eysholdt and Behrens [55]). The specified queries were transformed into executable data mining processes by means of code generation templates (Syriani et al. [155]). The code generation was performed with the *Epsilon* (Paige et al. [128]) 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:

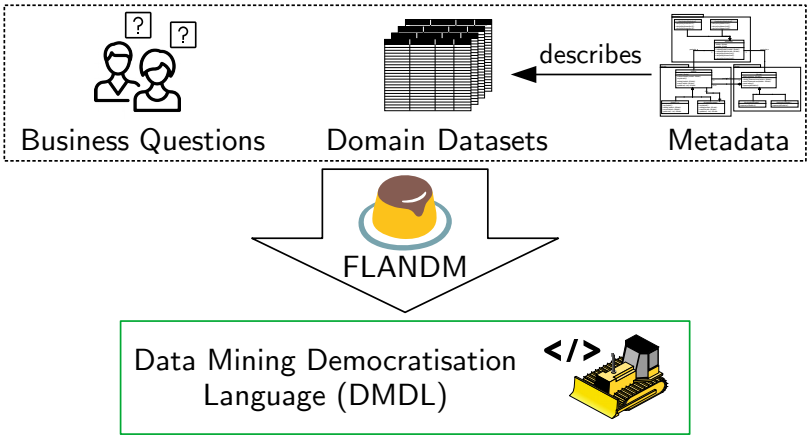


Fig. 3.2 Inputs and outputs of FLANDM's DMDL generation process.

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.

3.2 Overview of FLANDM

Here we give an overview of the elements involved in the creation of a new DMDL when using FLANDM. Figure 3.2 shows the general inputs and outputs of this framework, which we describe below.

There are three required inputs: (1) the set of *business questions* that the syntax of the resulting language should allow formulating; (2) a collection of *datasets*, i.e., two-dimensional data bundles, obtained from the domain data; and (3) some *metadata* about the provided datasets that, as we will see, are formalised as a model instance conforming to a specific metamodel.

From these inputs, the framework helps define a new DMDL adapted to the specified domain. The main benefit of this DMDL is that it can be used directly by domain experts. For that benefit to be true, the generated language offers the following features:

- An *easy-to-use syntax* to specify analysis queries. This syntax only contains high-level commands to indicate the analysis to perform; and vocabulary from the domain metadata, which allow selecting the data to be used in an analysis.
- An *execution system* for translating the specified query into a data mining process that tries to find an answer by studying the selected data. This translation happens in the background, freeing the user from having to know any low-level details of the involved data mining techniques.

The following sections describe how FLANDM helps in the definition of these two features of a DMDL. These descriptions use the diabetes analysis case study presented in Section 1.2.4 as running example.

3.3 Queries Specification

This section focuses on the development of a syntax and an editor for the specification of queries. We follow again a metamodeling-based approach (Kleppe [94]), 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 1.2.4, the entities would correspond to the patients diabetes data, and their attributes would be the gathered indicators and the final result of the test.

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

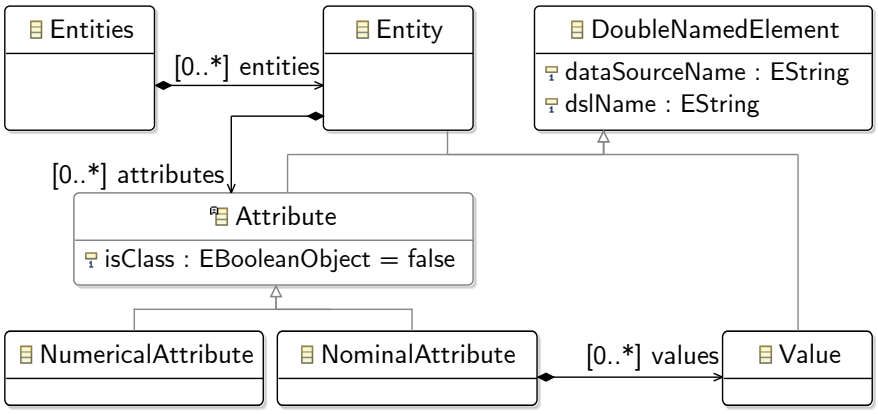


Fig. 3.3 Entities Metamodel

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. 3.3. 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.

Named elements in the entities metamodel 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

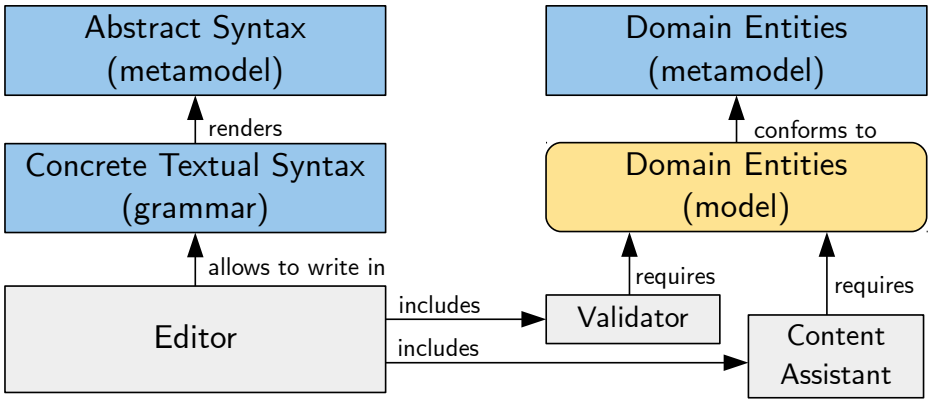


Fig. 3.4 Overview of FLANDM syntax and editor components.

Fig. 3.4 was obtained. It represents the components of FLANDM's query specification editor. Each one of these components are described in the following.

3.3.1 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 (e.g. Kang et al. [87], Čeh et al. [178]). FLANDM does not impose the use of any of these methodologies, therefore, DSL engineers might use their preferred one. In general, we have opted for using well-known techniques of Domain-Driven Design (Evans [54]) 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 work, 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 (Witten et al. [167]). However, we worked in a complementary language, Lavoisier, for providing non-experts with data selection mechanisms over high-level models of the domain information. This language is introduced in Chapter 4.

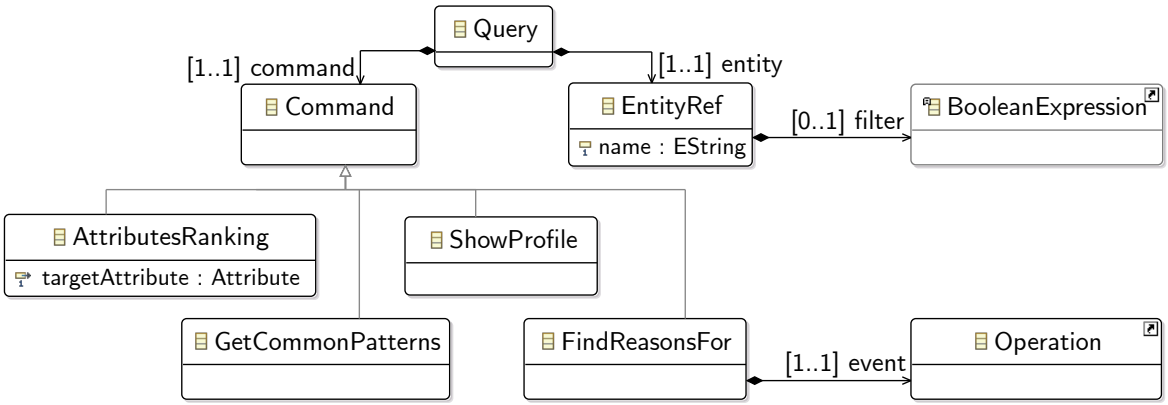


Fig. 3.5 Excerpt of the abstract syntax provided by FLANDM.

Table 3.1 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: analysed phenomena

3.3.2 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. 3.5 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 3.1 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.

3.3.3 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 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* [55], 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. 3.6 shows a validation function that

```
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. 3.6 Check function which validates the name of an entity.

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

$$\langle command \rangle ::= \langle showProfile \rangle \quad (2)$$
$$\mid \langle findReasonsFor \rangle \quad (3)$$
$$\mid \langle attributesRanking \rangle \quad (4)$$
$$\mid \langle getCommonPatterns \rangle \quad (5)$$

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

$$\langle showProfile \rangle ::= \text{`show\_profile'} \quad (7)$$

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

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

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

Fig. 3.7 Base grammar offered by FLANDM for the DMDLs.

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. 3.6, lines 02-04). If the name is not found, the validator raises an error with a proper explanation message (Fig. 3.6, 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.

3.3.4 Concrete Syntax

After defining the abstract syntax, a conforming concrete syntax must be provided. In FLANDM, a concrete textual syntax is defined through an Xtext grammar (Eysholdt and Behrens [55]). As with the abstract syntax, a base grammar provided as a base

for new DMDLs. Fig. 3.7 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 Figure 3.1).

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. 3.7 for the definition of their own concrete syntax when needed. It should be noticed that the grammar of Fig. 3.7 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 (Fister et al. [59]). The steps would be as follows:

- Provide new production rules, like the ones depicted in Fig. 3.7, 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.

3.3.5 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. 3.8 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

```
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. 3.8 Assistant function that suggests attributes of an entity.

for the entity in the model (Fig. 3.8, line 03), and then transverses its attributes list generating a proposal for each one of them (Fig. 3.8, lines 05-08).

3.4 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 [94], 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 3.1) 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.

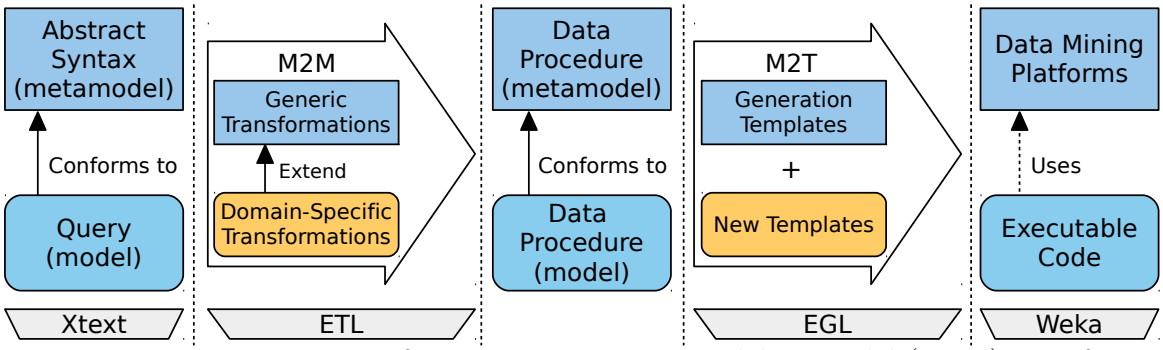


Fig. 3.9 Two-step query transformation process: a model-to-model (M2M) transformation is followed by a model-to-text (M2T) code generation phase.

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 (Pelleg and Moore [131]) clustering algorithm, whereas for other domains, other algorithms, such as DBSCAN (Ester et al. [53]), 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. 3.9. 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.

3.4.1 Data Procedure Metamodel

The metamodel diagram is shown in Fig. 3.10. A `Procedure` defines an abstract data mining task. All procedures operate over a `DataSource`, which is the equivalent to

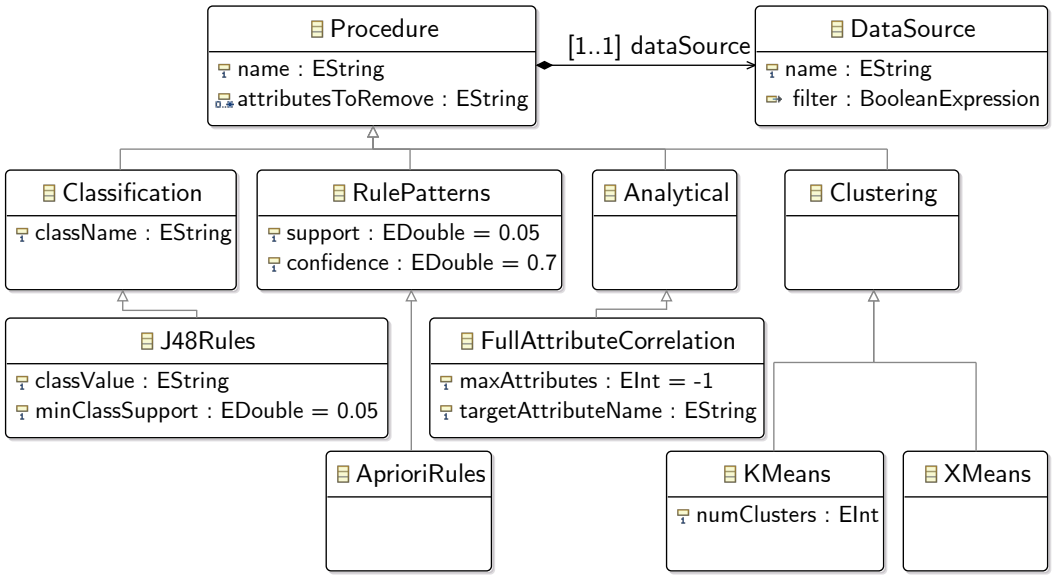


Fig. 3.10 Data procedure metamodel.

the `Entity` metaclass of the languages' abstract syntax metamodel (Fig. 3.5). This procedure metaclass is the root of a hierarchy that represents data mining methods. Intermediate metaclasses group 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 (Quinlan [135]) 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.

3.4.2 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. 3.9, 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)* (Kolovos et al. [99]). 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. 3.11 shows one of these base rules. In this case, according to its guard (Fig. 3.11, 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 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. 3.11.

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 (Campos et al. [28]). Nevertheless, this is a known issue in the data mining field (Wolpert and Macready [172]). 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.

3.4.3 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*

```
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;
   }
```

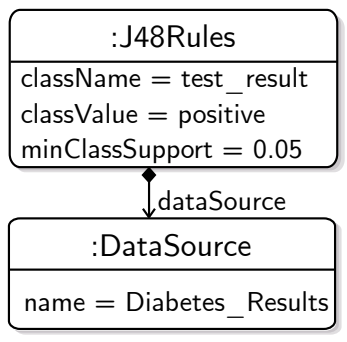


Fig. 3.11 Left: M2M transformation rule of a query in a J48Rules data procedure model; right: resulting J48Rules model of the M2M transformation over the example query.

Generation Language (EGL) (Rose et al. [141]). 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. 3.11, right can be seen in Fig. 3.12, 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 example, 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. 3.12, 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 which 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).

```
   // 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. 3.12 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.

In Fig. 3.12, 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.

3.5 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.

3.5.1 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 3.1).

2. A language for the medical domain, which analysed results of diabetes tests (Section 1.2.4).

3. A language for the business management domain, which analysed business reviews coming from the *Yelp*[1] 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 chapter (see Sections 3.1 and 1.2.4, 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[2] 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 [3] 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.

[1] https://www.yelp.com/
[2] https://www.yelp.com/dataset/challenge
[3] 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 equals denied
     of H-1B_Visas
     with year = 2017;
Q3 get_common_patterns
     of H-1B_Visas
     with status equals certified
       or status equals certified-withdrawn;
```

Fig. 3.13 Queries of the business reviews (Q1) and visa approval (Q2, Q3) DMDLs.

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 previously, Figure 3.13 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.

Table 3.2 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.

3.5.2 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.

Reusability Measurement Method

As a measure of component reusability, we relied on the cost productivity model provided by Gaffney and Durek [62]. 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 3.2.

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 $R = 0.2$.

$$C = (1 - R) \cdot 1 + R \cdot b = (b - 1) \cdot R + 1 \tag{3.1}$$

With these definitions, to calculate the value of C, Gaffney and Durek define the equation 3.1. 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.

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 (Boehm [22]).

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.

Table 3.3 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 %

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 3.3 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 3.3.

We based the calculation of b on three parameters, which are described in Table 3.4. *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

Table 3.4 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.

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.

$$b = \frac{MLOC + CLOC}{TLOC} \tag{3.2}$$

Considering these premises, b is defined as in equation 3.2. 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.

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 3.3 were used.

Table 3.5 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

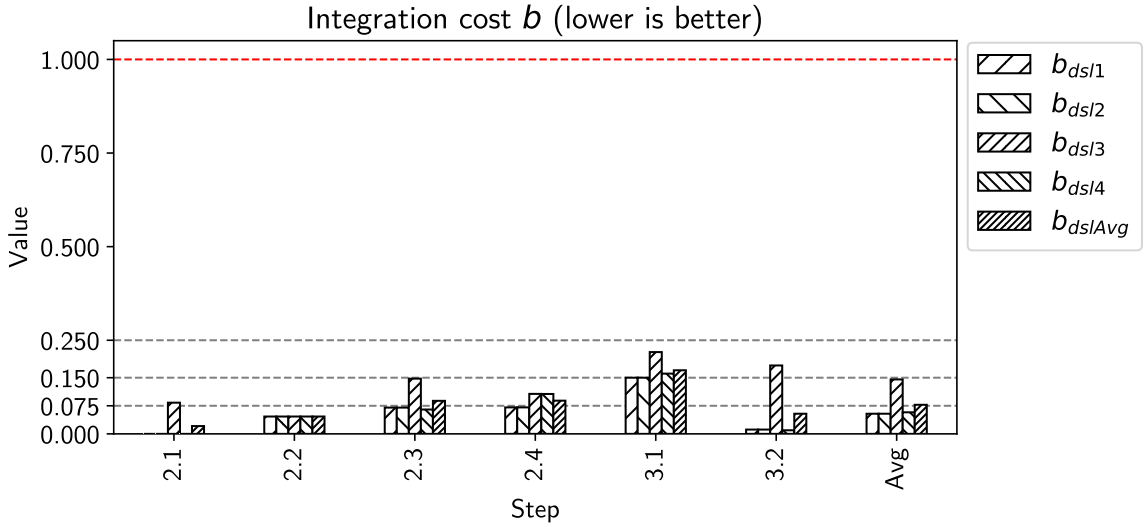


Fig. 3.14 Relative Integration cost (b) per development step of each implemented case study, plus its weighted average value (*Avg* b). The dashed line at value 1 specifies the critical point above which reutilisation is not cost-effective.

Reusability Results and Discussion

Table 3.5 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. Figures 3.14 and 3.15 depict these values graphically, so that they are easier to compare and visualise.

Figure 3.14 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 ($b_{DSL_4,S_{3.1}} = 16.1\%$; $b_{DSL_4,S_{3.2}} = 1\%$).

Figure 3.15 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

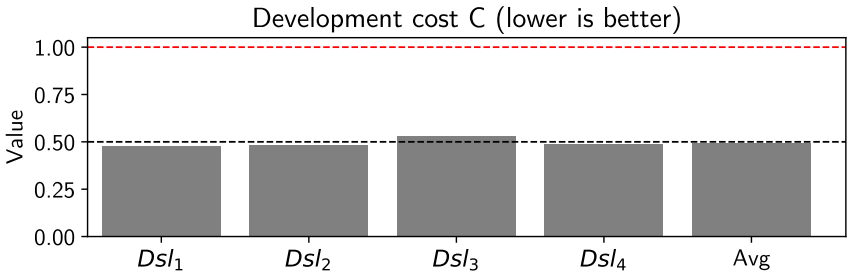


Fig. 3.15 Development cost C of the implemented case studies, along with the average value (Avg). Dashed line at value 1 marks the cost of creating each case study from scratch.

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. 3.16 shows the relation between R and C based on the Gaffney and Durek's cost formula [62], 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, 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

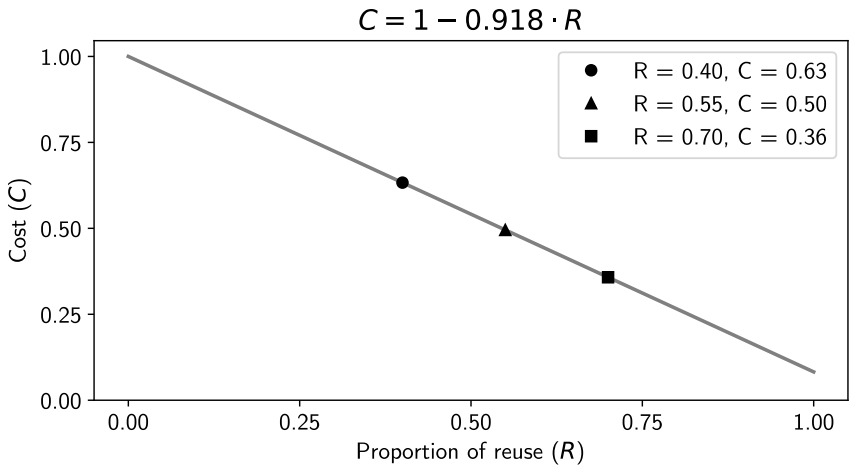


Fig. 3.16 Relation between DMDL cost C and proportion of reuse R, for $b = 8.2\%$.

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.

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.

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 (e.g. Adams et al. [2], Ajila and Wu [3], Johar et al. [81], Kim et al. [90]).

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 3.3) 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 3.3), 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:

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 3.3, 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 3.3). 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 $\backsim$50% as stated, then the reduction for this developer would be that 50%, although of a larger total effort.

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.

3.5.3 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 [47], 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 (Hinkel et al. [74], Kiczales et al. [89]).

Table 3.6 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

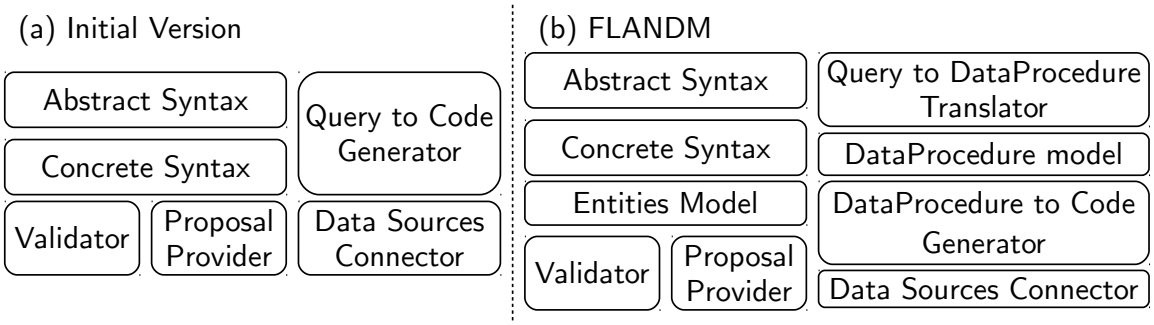


Fig. 3.17 DMDL architecture without (a) and with (b) FLANDM.

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 3.6. 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. 3.17. For simplicity, in the following, we refer to the architecture of the DMDL for the educational domain as the *initial version*.

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 3.3). 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.

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.

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.

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.

3.6 Summary

This chapter has presented FLANDM, a framework designed to reduce the development cost of Data Mining Democratisation Languages (DMDLs) that allows to adapt each defined language to the specific domain of application.

The framework was built over a previous work (de la Vega et al. [41]), where the architecture of our first DMDL was presented. For FLANDM, we refactored this initial architecture to support and ease component reuse, fixing some flaws detected in it. Since these DMDLs share several commonalities, the development cost reduction is achieved by means of a modular structure, which promotes component reuse and easy language customisation. The refactored infrastructure allows creating DMDLs 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. In addition, the modular structure of DMDLs generated with FLANDM allows for a better maintenance, as there is a minimum coupling between the different components.

Chapter 4

Lavoisier: High-Level Selection and Preparation of Data

4.1 Introduction

As described in Chapter 3.3, when creating a query with a DMDL, decision makers select (1) an analysis command to execute; and (2) the entity whose data is to be used in such analysis. Each DMDL has an entities model that contains the set of entities available for analysis. This set is static, so decision makers are constrained to select one of the available entities, not being possible to define their own customised ones.

The definition of new entities over which to invoke analysis processes is not a trivial task. Each entity is linked to a dataset, which stores the data used in the analysis. These datasets have to conform to a very specific two-dimensional tabular format, in order to be usable as input for data mining algorithms. In this format, all the information related to each one of the elements being analysed must be placed in a single row. For instance, if we are trying to find the reasons why a business becomes successful, all the information we have about a single business must be placed in a single row the dataset containing the data about businesses.

As a consequence, these algorithms cannot work with hierarchical or linked data such as JSON files, XML files or relational databases containing several tables with relationships between them. These are the formats in which data are typically stored or provided. Consequently, to execute a data mining algorithm, we need to transform data from their original representation to the commented tabular format that data mining algorithms can digest.

The creation of these datasets from the original data sources is often a tedious, highly technical, and prone to errors task. As such, this task is most of the time only

achievable by data scientists, which prevents average decision makers from creating custom datasets by themselves. Our systematic review detected that approaches that try to help these decision makers participate in this task are scarce, even when the inputs of a domain expert could play an important role in the success of an analysis.

To alleviate this situation, we present an operator that automates this task. Using this operator, we only need to specify which elements of the available data we want to include in an analysis. Then, the operator automatically computes the corresponding tabular representation. To execute this tasks, the operator executes a set of low-level data transformations, which adapts transformation patterns coming from different fields, such as *object-relational mappers* (Fowler [60], Hainaut [69]) or *data processing* (Cunningham [39], Wickham [165]).

To define this operator, we needed to determine how to represent the available data to analyse from each concrete domain. We opted for using object-oriented models to describe these data. Object-oriented models are nowadays widely used to construct *domain models* that can serve as a communication points between domain experts and developers (Evans [54]). Therefore, the use of domain models might make easier the participation of decision makers in the process of identifying relevant data of a particular domain to be analysed.

Therefore, this work focuses on the definition of an operator for automatically transforming data that can be accessed through an object-oriented representation into a tabular format that can be used as input for data mining algorithms. Moreover, based on the definition of this operator, we developed a language, called *Lavoisier*, which provides a set of high-level declarative primitives for constructing tabular datasets from object-oriented domain models. Using this language, data scientists and domain experts can focus on the selection of relevant data for an analysis and forget about the low-level details of the process required to transform the selected data, as this process is transparently executed thanks to our operator. Therefore, data scientists are relieved of executing them by hand, which decreases errors and increases productivity.

Expressiveness and effectiveness of our approach were assessed using a data mining open challenge belonging to the domain of business reviews (Yelp [174]). We specified different data selection and transformation processes over data of this challenge. Then, compared the results of performing these specifications with Lavoisier against two representatives of the state-of-the-art tools for data transformation, namely, the SQL language (Beighley [18]) and the Pandas data manipulation library (McKinney [114]). As a result of this comparison, we concluded that Lavoisier's dataset specifications are more compact, and require less constructs and with a higher level of abstraction.

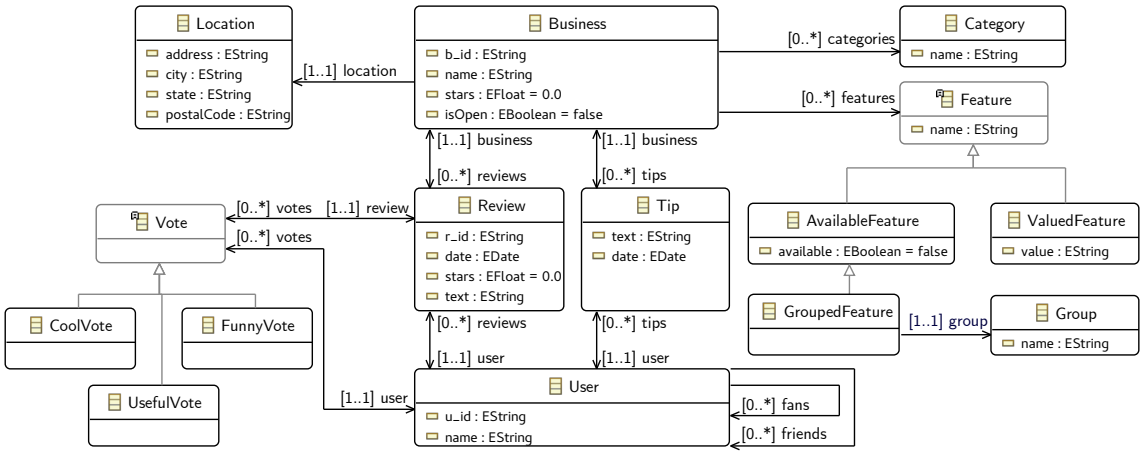


Fig. 4.1 Conceptual Model for the Yelp Dataset Challenge.

After this introduction, the work is structured as follows: Section 4.2 exposes the motivation behind this work. The chapter continues with a discussion about related work in Section 4.3. Our data transformation operator is described in Section 4.4. Section 4.5 presents the different features of the Lavoisier language. The chapter finishes with a recapitulation and an enumeration of future objectives of this work in Section 4.7.

4.2 Case Study and Problem Statement

This section describes the motivation behind the contributions of this chapter. First, a running example that we used throughout our descriptions is introduced. Then, we explain how our approach fits inside a generic data mining process. Finally, the motivation behind this work is detailed using the running example.

4.2.1 Running Example: The Yelp Dataset Challenges

Yelp is an American company that provides an online business review service. Using this service, owners can describe and advertise their businesses and customers can write their opinions about these businesses.

Yelp collects and makes available bundles of data for academic usage and proposes a new challenge periodically[1]. We used these data and challenges as running example throughout this article.

[1] https://www.yelp.com/dataset/challenge

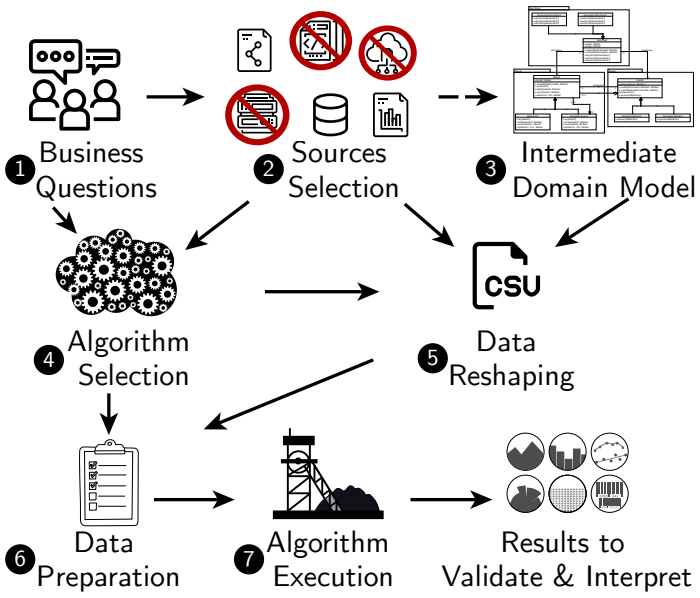


Fig. 4.2 Data mining process. (Some icons by Smartline from Flaticon)

Yelp provides their data as a bundle of interconnected JSON (*JavaScript Object Notation*) files. To help visualise these files, we rebuilt the conceptual object-oriented model to which these files would conform. This conceptual data model is depicted in Figure 4.1 .

As it can be seen, for each registered business, Yelp provides information about its location; the different features it offers, such as the availability of Wi-Fi or a smoking area; and the categories which best describes it, e.g., Cafes, Restaurant, Italian, Gluten-Free, and so on. Users can *review* these businesses, rate them and introduce a text describing their experience. Additionally, users can write *tips*, which are small pieces of advice about a business, such as *do not miss its salmon!* Yelp also provides some social network capabilities, so users can have *friends* or *fans*. Users can also receive *votes* in their reviews in case other users found these reviews *funny*, *useful* or *cool*.

Using these data, *Yelp* proposes as challenges analysis tasks like identifying reasons behind a business becoming successful, or finding what kind of opinions are most likely to set a new trend.

4.2.2 Data Mining Processes Extended

In this section, some stages of the data mining process introduced in Chapter 1.3 have been split into several substages, in order to clarify some issues related to the work of this chapter.

Any data mining process is created to answer business questions (Figure 4.2, Step 1). These are derived from the business or domain experts needs. For instance, in the case of Yelp, data mining processes are elaborated to try to answer the questions raised in their challenges, such as finding reasons that make a business successful.

Once the business questions are identified, we need to decide what data sources will be used to answer them (Figure 4.2, Step 2). These sources are also selected with the help of the business experts. We might use several data sources, each one of a different nature, for answering the business questions. Yelp challenges' data include information from different subsystems, such as the review system or its social network.

When several data sources are used to answer business questions, the elaboration of an *intermediate domain model* that abstracts from these sources might be helpful (Figure 4.2, Step 3). This data store would collect the data available in a particular context for being used as input for data mining algorithms. This *intermediate data store* might be skipped in some cases, such as, for example, when a single data source is used. In the case of Yelp, this intermediate data store is provided by Yelp itself by means of a set of interconnected JSON files. As previously commented, these JSON files can be abstracted into the conceptual data model depicted in Figure 4.1.

Subsequently, we must select the data mining technique that we consider most adequate for answering each business question (Figure 4.2, Step 4). For instance, in Yelp challenges, *clustering techniques* might be employed to group businesses according to the similarity of their characteristics. This might give indications of what commonalities are shared among successful and unsuccessful businesses.

For each data mining technique, such as *clustering*, there is a plethora of algorithms available in the literature. Each one of these algorithms is designed to perform better than the others depending on certain characteristics of the input data. Therefore, we are also in charge of selecting the algorithm that best fits with the nature of our input data. For instance, to analyse data from Yelp, an algorithm like DBSCAN (Ester et al. [53]) might be a reasonable choice for reasons that are beyond the scope of this work.

Most data mining algorithms can only accept as input data arranged in a very specific tabular format. Data scientists often refer to bundles of data arranged in this format as *datasets*. Therefore, as next step of a data mining process, we need to reshape the data contained in the *intermediate data store* - or the data sources when this model is skipped - to create *datasets* that can be digested by data mining algorithms (Figure 4.2, Step 5). The work presented in this chapter focus on this specific step of a data mining process. Consequently, this step is explained more in detail in next section.

(a) Information of each business in a single row

BName	Stars	WiFi	Parking
Pete's Pizza	4.5	true	true
Sushi & Go	3.8	false	true
Wine Heaven	4.0	true	

(b) Information of each business in several rows

BName	Stars	Feature	Available
Pete's Pizza	4.5	WiFi	true
Pete's Pizza	4.5	Parking	false
Sushi & Go	3.8	WiFi	false
...	...	...	....

Fig. 4.3 Two tabular arrangements of businesses' data.

In addition to this reshaping, each algorithm might impose other extra constraints to their input data. For instance, some distance-based algorithms require data to be normalised into the range [0, 1]. Therefore, we would need to perform some extra data transformations in order to ensure these constraints are satisfied before using a dataset as input data for these algorithms. These transformations can take place after a dataset has been produced or at the same time it is generated (Figure 4.2, Step 6).

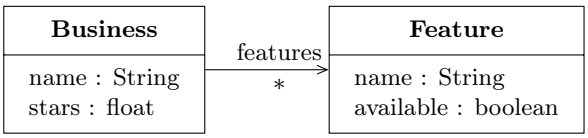
Finally, we execute the selected data mining algorithms with the generated datasets (Figure 4.2, Step 7), which would produce several results. These results must be analysed to asses their quality and reliability. Then, curated results can be passed to the business or domain experts, who would interpret them to extract some conclusions and make some decisions. As an example, after performing an analysis over data from a Yelp challenge, the obtained results might provide interesting advice for an *entrepreneur* before starting a new venture.

Next section details the stage of this data mining process in which this work focuses: the creation of tabular datasets from non-tabular information, such as linked or hierarchical data.

4.2.3 The Data Reformatting Problem

Before executing a data mining algorithm, we need to transform the data to be analysed into a specific tabular format. This format, in addition to being tabular, imposes an extra and non-trivial constraint: for each instance of the domain entity being analysed, all the information about this instance that we want to include in an analysis must be placed in a single row of the tabular format. In the following, we elaborate on this issue to clarify it.

a) Domain Model



b) Model Instances

```
[ {"name" : "Pete's Pizza", "stars" : 4.5,
   "features" : [
        {"name" : "WiFi", "available" : true},
        {"name" : "Parking", "available" : true}]},
  {"name" : "Sushi & Go", "stars" : 3.8,
   "features" : [
        {"name" : "WiFi", "available" : false},
        {"name" : "Parking", "available" : true}]},
  {"name" : "Wine Heaven", "stars" : 4.0,
   "features" : [
        {"name" : "WiFi", "available" : true}]} ]
```

Fig. 4.4 (a) Business ratings model excerpt; (b) graph with some instances of (a).

Let us suppose that, in the context of the running example, we want to identify business features, or combinations of features, that might lead to a business having a high stars rating. In this case, businesses would be the domain entity being analysed. To compute this information, we decided to use as information for the analysis the business name to identify businesses, its stars rating, and its set of available features. In this case, our problem is how to produce a tabular arrangement like illustrated in Figure 4.3 (a). As it can be seen, in this arrangement, all the information related to one business is placed in just one row. Alternative tabular representations, such as shown in Figure 4.3 (b) would not be valid inputs to nowadays existing data mining algorithms for the analysis of businesses.

Therefore, we must face the problem of how to produce an adequate tabular representation as depicted in Figure 4.3 (a). The very first to do to achieve this goal is to filter the original data so that it only contains those data in which we are interested on. This can be easily achieved using data management tools. Figure 4.4 shows an excerpt, in JSON format, of the results of this filtering process.

The second task involves rearranging these data into the specific tabular format of Figure 4.3 (a). This is not a trivial task and it might involve a lot of small issues and picky details, as we comment throughout this article. In the specific case of Figure 4.4, a question to be addressed is that JSON data is hierarchical. This is, a JSON object can contain inside another JSON object that might containt another JSON object, and

so on. Therefore, we need to *flat* this hierarchy to convert it into a an information vector.

In this particular case, we must combine the information belonging to the *Business* and *Feature* classes so that it becomes a flat data vector. To do it, we would create a new column for each feature that a business might have, e.g. *WiFi*. Each column would take as value *true* or *false* depending on the registered value for that feature. Other cases might need different strategies.

Generally speaking, our problem is that domain data is often stored in a hierarchical or linked form. For instance, in the case of JSON files, as already seen, an object can contain another object. When using relational databases, a table might reference another table by means of a foreign key. Therefore, we need to convert these hierarchial and linked data into flat vectors of information. This process will be referred in the following as a *flattening operation*[2]. Next section analyses how this flattening operation can be achieved using state-of-the-art techniques.

4.3 State-of-the-Art Data Flattening Strategies

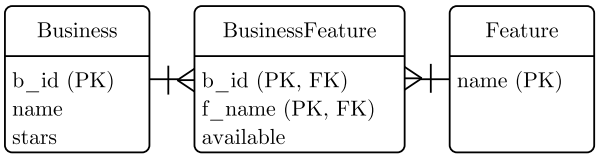
To address the *data formatting problem* presented in the previous section, data scientists rely currently on data management languages or libraries, such as *SQL (Structured Query Language)* (Beighley [18]) or *Pandas* (McKinney [114]). This section reviews these state-of-the-art strategies, highlighting strengths and weaknesses.

4.3.1 SQL Languages

SQL (Structured Query Language) is probably the most well-known data management language, which was created to manipulate relational database tables. Using SQL, the contents of two or more entities, represented as relational tables, can be combined by means of *join* operations. A join operation takes two tables and one column of each table as input, calculates the Cartesian product of these tables and removes all those tuples where the specified pair of columns do not match, providing a new table as a result. Therefore, a join operation can be used to merge two tables into a single table, which is somehow what we are looking for. Nevertheless, joins do not fulfill the requirement of a flattening operation, which is placing all the information about a single entity in a same row.

[2]This term is inspired by a similar operation often performed in functional programming

a) Relational tables



b) Join Result

BusinessName	BusinessStars	featureName	available
Pete's Pizza	4.5	WiFi	true
Pete's Pizza	4.5	Parking	true
Sushi & Go	3.8	WiFi	false
Sushi & Go	3.8	Parking	true
Wine Heaven	4	WiFi	true

Fig. 4.5 (a) Business and Features entities represented as relational database tables; (b) Result of a join operation between Business and BusinessFeature tables.

To illustrate this shortcoming, Figure 4.5 (a) shows the `Business` and `Feature` entities represented as relational database tables, whereas Figure 4.5 (b) shows the result of a join between these tables, using *Business.b_id* and *BusinessFeature.b_id* as joining columns. As it can be seen, in the result table, information about individual business is scattered across several rows. To address this issue, several workarounds might be used, as we comment in the following.

Listing 4.1 SQL flattening operation using aggregation queries and the case operator.

```
1  select b.b_id, b.name, b.stars,
2         max(case features.name when 'Parking'
3             then available end) as feature_parking,
4         max(case features.name when 'WiFi'
5             then available end) as feature_wifi,
6         max(case features.name when 'Smoking'
7             then available end) as feature_smoking
8  from features
9  right join business b on b.b_id = features.b_id
10 group by b.b_id, b.name, b.stars
```

Listing 4.1 shows an example where, by employing SQL's *CASE* operator in an aggregation query, the tabular representation of Figure 4.3 (a) can be obtained from the tables depicted in Figure 4.5 (a). As it can be seen, a main shortcoming of this strategy is that we need to add an aggregation query with a case operator for each

column of the resulting table we want to generate. This strategy might be prohibitive when processing a larger number of features. For instance, if we wanted to include information about business benefits, disaggregate per month and including data of the last five years, we would need to create a SQL query with 60 nested aggregation queries. Moreover, as time passes, we would need to update this script to add aggregation queries for new months and years, as well as removing obsolete months and years, if required.

4.3.2 Data Warehouse Operations

Data warehouses store large volumes of information, which can be consulted for reporting and analytical purposes (Malinowski and Zimanyi [112]). These data are usually organised into multidimensional models based in *facts* and *dimensions*. *Facts* store *quantitative* measures around a business concept, while dimensions offer different perspectives, such as space and time, from which to obtain and analyse fact measures.

Data warehouse solutions typically provide special systems known as *OLAP (Online Analytical Processing)* (Wrembel and Koncilia [173]) to query and analyse data from their multidimensional models. These systems include operators for data selection (*slice* and *dice*), aggregation (*roll-up*, *drill-down*), and rotation (*pivot*). From these, the most interesting one for our objective is the *pivot* operator.

By performing a pivot operation, we can compact into a single row information about the same entity that is initially scattered across several rows of a table. This is, using a pivot, we can transform the table of Figure 4.5 (b) into Table 4.3 (a). In this case, the pivot operation would take as input the table of Figure 4.5 (b), a *pivoting column*, that would be *featureName*, and a set of columns to be pivoted, that in this case would be just *{available}*. With these inputs, the pivot operation would work as follows.

First, the structure of the output table is determined. To do it, all columns not involved in the pivot operation are added as columns to the output table. In our case, *BusinessName* and *BusinessStars* would be added. Then, for each distinct value of the pivoting columns, a new column is added to the output table. In our example, the pivoting column is *featureName*, which has as values *WiFI* and *Parking*. So, the columns *WiFI* and *Parking* are added to the output table.

Once this table structure is created, the output table is populated with data from the input table. Each distinct tuple for the non-pivoted columns is added as a new row in the output table. In our example, the tuples *(Pete's Pizza, 4.5)*, *(Sushi & Go, 3.8)*, and *(Wine Heaven, 4.0)* would be added as new rows to the output table. As it

can be noticed, these rows are incomplete and the newly created columns, i.e., *WiFI* and *Parking*, would still need to be filled. To fill these columns, the pivot operator checks, for each new added row, whether the input table contains a row that has the non-pivoted values plus the corresponding pivoting column value. If such a row is found, the pivot operator copies the value of the pivoted column of this row into the corresponding cell of the output table. For instance, continuing with our example, to calculate the value of the *WiFi* column for the *(Pete's Pizza, 4.5)* row, the pivot operator checks if the input table has a row containing the values *(Pete's Pizza, 4.5, WiFi)*. If so, the value of the *available* column for that row is copied into the cell corresponding to the *WiFi* column. Finally, it is worth to comment that this operator has associated a set of picky low-level details, such as how to proceed when no row is found in the input table for a pivoted column; or when, oppositely, several rows are found. These cases are not commented in this section for the sake of simplicity, since our goal is just to illustrate how a pivot works. A more formal definition of this operator is given in Section 4.3.2.

This operator, apart from being present in OLAP systems, can also be found with slightly different implementations in some proprietary database management systems such as SQL Server (Cunningham [39]), and in popular data analysis frameworks like R (Wickham [165]).

As it can be seen, using a pivot, we can compact the table of Figure 4.5 (b) into an appropriate tabular format. This means that by combining *joins* and *pivots*, we could perform a flattening operation. Nevertheless, as the information to be included in an analysis grows, the number of chained *join* and *pivot* operations increases, which can lead to large and complex data manipulation scripts, that would be hard to update. This kind of complexity is what we try to reduce by creating a flatten operator that performs all these operations transparently.

4.3.3 Data Management Frameworks and Libraries

Some frameworks for data analysis, such as *R* [157], provide their own data management languages. On the other hand, there are libraries for general purpose programming languages, e.g., *Pandas* (McKinney [114]) in the case of Python. Both approaches offer facilities for retrieving, cleaning and formatting data. Nevertheless, these approaches do not include advanced operators, different from the previously commented ones, that can be used to perform a *flatten* operation. So, as with previous approaches, users of these frameworks and libraries have to manually combine low-level operators to produce the required tabular structures.

In summary, as it can be seen, state-of-the-art approaches for data manipulation do not provide any high-level operator for the arranging data belonging to different entities into a tabular data structure that fulfils the requirements demanded by data mining algorithms. To improve this situation, we have created a new high-level *flattenning* operator, which is described in the next section.

4.3.4 Automatic Feature Extraction

The previous approaches allow data scientists to manually perform flattening operations. However, there is a research topic known as *propositionalisation* (Boullé et al. [23], Kanter and Veeramachaneni [88], Knobbe et al. [96], Samorani [142]), whose aim is to automatically reduce multi-relational data to a single-table structure for analysis purposes, i.e., to automate the flattening process. This automation works as follows: starting from the entity of interest, e.g., Business of the Yelp case study, an algorithm randomly generates features by applying aggregation functions over the relationships between the selected entity and other entities in the model.

This random exploration has the potential to discover previously unknown features that are relevant for an analysis. However, it also has some drawbacks, such as scalability (the combination of applicable aggregations with the possible relationship transversals forms an enormous feature space to explore) or performance (i.e. most obtained features may not be useful at all, which makes necessary to generate a large number of them). In addition, this approach only works by aggregating relations of the conceptual model, and the result of each aggregation is a single value. Therefore, flattening scenarios where the reformatted information is distributed into several columns, such as the features one presented in Figure 4.3 (a) would not be possible when applying this approach.

4.4 Flattenning Operator Description

This section provides the specification of a flattening operator that takes as input several entities and their associated data, and produces as output a tabular representation of these inputs that can be digested by data mining algorithms. Before specifying this operator, we need to define the format in which input entities should be provided. This is, we need to define a notation or language for the *intermediate domain model* (Figure 4.2, step 3). Typically, object-oriented models (Evans [54]), ontologies ([145] or multidimensional models (Golfarelli and Rizzi [66]) are used for this purpose. Each one

of these options provides advantages and disadvantages, depending on their intended use.

In this work, we focused on object-oriented domain models because of two reasons. Firstly, this kind of models are a good starting point for the long-term goal of defining flattening operators for any kind of domain model. Once this first operator is available, techniques and strategies developed for its creation might be used as basis for the definition of new operators, for instance, for ontologies. Secondly, object-oriented domain models have been acknowledged in the literature as an effective mechanism to improve the communication between stakeholders and developers (Evans [54]). As an example, these models can contribute to the incorporation of domain experts into software development processes. These domain experts, as commented in Section 4.2, are key to identify the data that might be of relevance for each concrete analysis process. Therefore, we expect that, by using object-oriented models, domain experts can participate more directly in the data selection tasks of these processes.

The strategy we have used to define this flattening operator for object-oriented models is as follows: there is a trivial flattening case that involves transforming a single class, without multivalued attributes, into a tabular representation. In this case, data would be tabulated according to our requirements by default, so no action would be needed. Therefore, if we were able to reduce all potential scenarios that might appear in an object-oriented model to this trivial case, the flattening problem would be solved. So, we have focused on determining how to reduce each one of these scenarios into a single class by means of chaining data transformations. This single class must be the one that represents the entities under analysis, e.g., businesses or reviews. In the following, this prominent single class is referred as the *main class*.

Thus, an implementation of the flattening operator should identify each one of these scenarios in the input model and execute the corresponding data transformations until reducing the original model to the trivial case. These data transformations are based on typical data management operations, such as *joins* or *pivots*, as well as on transformation patterns used by *Object-Relational Mappers* (Atzeni et al. [8], Fowler [60], Hainaut [69]).

Next subsections describe and formalise how each one of these reductions work. Each definition and pattern is explained with the help of the Yelp case study.

4.4.1 Preliminaries

Before explaining how our reduction patterns work, we specify the formal notation we used to define them.

Primitive Types

Definition 4.4.1. *A **primitive type** T is a set of well-defined values.*

Definition 4.4.2. *An **literal value** of a primitive type T is an element lv where $lv \in T$.*

Examples of primitive types are *Integer*, *Char*, *Boolean* or *String*. These types are usually offered as built-in types in programming languages. Examples of literal values of these types are 17, 'c', *true* or "John", respectively.

In the following, when a name appears in italics and with the initial letter capitalised, it represents the set of all elements denoted by that word. For instance, *PrimitiveType* represents the set of all primitive types, and *Type* is the set of all existing types.

Object-Oriented Models

Definition 4.4.3. *A **class** C is defined as a tuple $(n, Properties) \in String \times Set\{Property\}$, where n is the name of the class, and $Prop$ is the set of properties of the class. Classes also define types, i.e., $C \in Type$.*

Definition 4.4.4. *A **property** is defined as a tuple $(n, t, c) \in String \times Type \times Cardinality$, where n is the name of the property, t is a type (which can be a primitive type or a class type), and c is a cardinality.*

Definition 4.4.5. *A **cardinality** $c \in \mathbb{N} \times \mathbb{N} \cup \{*\}$ is a pair of values (c_{min}, c_{max}) that determine the number of values that a property of a class instance can hold. If $c_{max} \in \mathbb{N}$, then $c_{max} \geq 1 \wedge c_{min} \leq c_{max}$. Also, c_{max} can take a special value, $*$, which represents the absence of a limit, i.e., an instance can have an unbounded number of values for a property p when $c_{max} = *$.*

For example, if $c = (1, 3)$ for certain property $p = (n, t, c)$ of a class C, one value of p has to be defined at least in each instance of C, and, at most, p can have 3 values.

Definition 4.4.6. *An **attribute** is a property (n, t, c) where $t \in PrimitiveType$.*

Definition 4.4.7. *A **reference** is a property (n, t, c) where $t \in Class$.*

Definition 4.4.8. ***typeOf*** $: (LiteralValue \cup Property) \rightarrow Type$ *is a function that,*

1. *given a literal value lv, returns the primitive type to which that value belongs. This is, $typeOf(lv) = t \iff lv \in t$.*
2. *given a property $p(n, t, c)$, $typeOf(p) = t$.*

Class Instances

Definition 4.4.9. *Given a class* $C = (n, Prop)$*, with* $Prop = \{p_1, \cdots, p_n\}$*,* $p_i = (n_i, t_i, (c_{min_i}, c_{max_i}))$*, an* ***instance*** *is a tuple* $(values_1, \cdots, values_n)$ *where* $values_i$ *is a tuple with* $c_{min_i} \leq |values_i| \leq c_{max_i}$*,* $typeOf(v) = t_i, \forall v \in values_i$*.*

Definition 4.4.10. ***propValues*** $: Instance \times Property \rightarrow Tuple\{LiteralValue\} \cup Tuple\{Instance\}$ *is a function that, given an instance* $ins = (values_1, \cdots, values_n)$ *of a class* $C = (n, Prop)$ *with* $Prop = \{p_1, \cdots, p_n\}$*, then* $propValues(ins, p_i) = values_i$*.*

Instances might have undefined properties. If a property p_i is not defined in an instance, then $values_i = ()$, which is an empty tuple. Obviously, $c_{min_i} = 0$ is required to accept instances with zero values in a property. This is equivalent to p_i being *null*, as usually denoted in conventional programming languages.

Definition 4.4.11. *A* ***data bundle*** *is defined as a tuple* $(DB_{class}, DB_{data}) \in Class \times Set\{Instance\}$*, where* $DB_{class} \in Class$*, and* DB_{data} *is a set of instances of* DB_{class}*.*

Tables

Definition 4.4.12. *A* ***table*** T *is a tuple* (T_{header}, T_{data})*, where* T_{header} *is also a tuple* $(columnName_1, \cdots, columnName_n)$*, and* T_{data} *is a* $m \times n$ *matrix of literal values, with* $typeOf(T_{data}(i, j)) = typeOf(T_{data}(i, k)), \forall i \in [1, m]$*,* $\forall j, k \in [1, n]$*, i.e., all values on the same column are of the same type.*

Auxiliary Functions

Definition 4.4.13. ***stringify*** $: Tuple\{LiteralValue \cup Instance\} \rightarrow String$ *is a function that, given a tuple of elements, produces a string of characters by concatenating the representation of each one of these elements as a* String*. Concatenated values are separated by the char '_'.*

For instance, *stringify("John", 27, true)* would return *"John_27_true"*.

Definition 4.4.14. *The* ***tuple concatenator****, denoted as* $\bullet$*, is a function* $\bullet : Tuple \times Tuple \rightarrow Tuple$ *that, given two tuples* $t_a = (t_{a_1}, \cdots, t_{a_n})$ *and* $t_b = (t_{b_1}, \cdots, t_{b_m})$*, produces as a result a tuple* $t_{ab} = (t_{a_1}, \cdots, t_{a_n}, t_{b_1}, \cdots, t_{b_m}))$*.*

Definition 4.4.15. ***emptyTuple*** $: \mathbb{N} \rightarrow Tuple$ *is a function that, given a number* $n \geq 1$*, returns a tuple* $t = (t_1, \cdots, t_n)$ *where* $t_i = (), \forall i \in [1, n]$*.*

Definition 4.4.16. ***projection*** : $Instance \times Set\{Property\} \rightarrow Tuple$ *is a function that, given an instance* ins *of class* $C = (n, Prop)$ *and a set of properties* $ProjectionProp = \{pp_1, \cdots, pp_n\}$, $ProjectionProp \subset Prop$, *then the expression* $projection(ins, ProjectionProp) = ins'$, *where* ins' *is a tuple of elements calculated as* $ins' = (propValues(ins, pp_1), \cdots, propValues(ins, pp_n))$.

Definition 4.4.17. ***filterEqual*** *is a function* : $Set\{Instance\} \times Set\{Property\} \times Tuple \rightarrow Set\{Instance\}$, *that, given a set of instances* Ins *of a class* C *(n, Prop), a set of properties* $FilterProps = \{fp_1, \cdots, fp_n\}, FilterProps \subset Prop$, *and a tuple of values* $FilterValues = (values_1, \cdots, values_n)$, *then*
$filterEqual(Ins, FilterProps, FilterValues) =$
$\{ins \mid ins \in Ins \wedge propValues(ins, fp_i) = values_i, \forall i \in [1, n]\}$.

4.4.2 Basic Transformation Operations

Here we define two basic data transformation operations, *join* and *pivot*, that we used to define the flattening operator.

Join

The object-oriented *join* operator combines instances of a data bundle of a class C with instances of one of its references, producing a new data bundle. As compared to the classical natural join, this operation can be considered as a *left outer join*, as we are interested in conserving all instances of the class C even in those cases where the processed reference is empty. Algorithm 1 describes precisely how this concrete join operation is computed.

A *join* accepts as input a data bundle $A(A_{class}, A_{data})$ and a reference *ref*, which belongs to A_{class} and has as type another class B. With these inputs, the join operator produces as output a new data bundle A', whose class has all the properties of A_{class}, minus the reference *ref*, and all the properties of B. As data, the output data bundle has all instances of A_{class} that have no relation with any instance of B, and, for each instance of A_{class} related to one or more instances of B, the instances resulting of combining such an instance of A_{class} with each instance of B to which the A_{class} instance relates.

As an example, if we perform the operation *join(Business, features)* over the data bundle of Figure 4.4, we would get a data bundle whose instances would have the same structure as the table shown in Figure 4.3 (b).

Algorithm 1: Join operation.

Input: A data bundle $A = (A_{class}, A_{data})$, $A_{class} = (name_A, Prop_A)$
Input: A reference ref
Precondition: ref is a reference, $ref \in Prop_A$
Output: A data bundle $A' = (A'_{class}, A'_{data})$
// Construct A' class
Let $B \leftarrow typeOf(ref), B = (name_B, Prop_B)$;
$name' \leftarrow stringify(name_A, \text{“}and\text{”}, name_B)$;
$Prop' \leftarrow Prop_A - \{ref\} \cup Prop_B$;
$A'_{class} \leftarrow (name', Prop')$;
// Construct A' data
$A'_{data} \leftarrow \emptyset$;
foreach $a \in A_{data}$ **do**
  case $propValues(a, ref) = ()$ **do**
    $A'_{data} \leftarrow A'_{data} \cup \{projection(a, Prop_A - \{ref\}) \bullet emptyTuples(q)\}$ where $q = |Prop_B|$;
  case $propValues(a, ref) = BInstances_a$ **do**
    foreach $b \in BInstances_a$ **do**
      $A'_{data} \leftarrow A'_{data} \cup \{projection(a, Prop_A - \{ref\}) \bullet b\}$;
    end
  end
end

FinancialResults
businessName
year
month
week
in
out

businessName	year	month	week	in	out
Pete's Pizza	2018	Jan	1	2000$	1500$
Pete's Pizza	2018	Jan	2	1450$	1200$
Pete's Pizza	...	...	...	...	...
Pete's Pizza	2019	Jan	2	1700$	1350$
Pete's Pizza	...	...	...	...	...
Sushi & Go	2018	Jan	1	4000$	2200$

Fig. 4.6 A data bundle *FinancialResults* to be pivoted. Left, the class of the bundle; right: class instances represented in a table

Pivot

Algorithms 2, 3, and 4 describe precisely how the pivot operator works. This operator was informally introduced in Section 4.3. To avoid overwhelming the reader with intricate low-level details, we have considered in these algorithms that properties of a class can hold just single values, i.e, their max cardinality is 1. The same assumption is made throughout this section, and, at the end of it, we explain how this operator can be easily generalised to work with multi-valued properties.

Algorithm 2: Pivot operation.

Input: A data bundle $A = (A_{class}, A_{data}), A_{class} = (name_A, Prop_A)$,
Input: A set of static properties $SP = \{sp_1, \cdots, sp_p\}$
Input: A set of pivoting properties $Pivoting = \{ping_1, \cdots, ping_q\}$
Input: A set of pivoted properties $Pivoted = \{ped_1, \cdots, ped_r\}$
Input: A set of aggregations $Aggregates = \{agg_1, \cdots, agg_r\}$
Precondition: $SP \neq \emptyset \wedge SP \subset Prop_A$
Precondition: $Pivoting \neq \emptyset \wedge Pivoting \subset Prop_A$
Precondition: $Pivoted \neq \emptyset \wedge Pivoted \subset Prop_A$
Precondition: SP, $Pivoting$, and $Pivoted$ are disjoint sets.
Precondition: $agg_i : Tuple(typeOf(ped_i)) \rightarrow T_i$, $T_i \in Type$
Output: A data bundle $A' = (A'_{class}, A'_{data})$

```
1 Let aggregateOf : Pivoted → Aggregate | aggregateOf(ped_i) = agg_i
2 A'_class, originalProperty, NewProperties ← ConstructClass()
3 A'_data ← ConstructData()
```

Algorithm 3: Pivot - *ConstructClass* Function.

Input: Those of the pivot operation.
Output: $A'_{class} = (name_{A'}, Prop_{A'})$
Output: A function $originalProperty : Property \rightarrow Property$
Output: A $Set\{Property\}$ $NewProperties$

```
1  Function ConstructClass:
2      Prefixes ← {stringify(projection(ins, Pivoting)) | ins ∈ A_data};
3      NewProperties ← ∅;
4      foreach ped = (name, type, c) ∈ Pivoted do
5          foreach prefix ∈ Prefixes do
6              newName ← stringify((prefix, name));
7              newProperty ← (newName, typeOf(aggregateOf(ped)), c);
8              originalProperty(newProperty) ← ped;
9              NewProperties ← NewProperties ∪ {newProperty};
10         end
11     end
12     name_A' = stringify((name_A,''));
13     Prop_A' = SP ∪ NewProperties;
```

The pivot operation accepts as input a data bundle A, three disjoint subsets of their properties, and a set of aggregation functions. The subsets are called *static properties*, *pivoting properties*, and *pivoted properties*, and they represent, respectively, the properties that are not affected by the pivot operation; the properties that are used as values to pivot; and the properties that are pivoted, i.e., rearranged. There must be

Algorithm 4: Pivot - *ConstructData* Function

Input: Those of the pivot operation.
Input: $A'_{class} = (name_{A'}, Prop_{A'})$
Input: A function $originalProperty : Property \rightarrow Property$
Input: A $Set\{Property\}$ $NewProperties$
Output: A $Set\{Instance\}$ A'_{data}

```
1  Function ConstructClassData:
2      A'_data ← ∅, PartialIns ← {projection(ins, SP) | ins ∈ A_data};
3      foreach partialIns ∈ PartialIns do
4          newValues ← ();
5          Tuples ← filterEqual(A_data, SP, partialIns);
6          foreach newProp ∈ NewProperties do
7              origProp ← originalProperty(newProp);
8              origValues ← (propValues(t, origProp)|t ∈ Tuples);
9              if Set(origValues) = {()} then
10                 aggValue ← ();
11             else
12                 aggFun ← aggregateOf(origProp);
13                 aggValue ← aggFun(origValues);
14             newValues ← newValues • (aggValue);
15         end
16         newIns ← partialIns • newValues, A'_data ← A'_data ∪ {newIns};
17     end
```

FinancialResults'
businessName
2018_Jan_in
2018_Jan_out
2018_Feb_in
2018_Feb_out
2019...

businessName	2018 Jan in	2018 Jan out	2018 Feb in	2018 Feb out	2019 ....
Pete's Pizza	8500$	5350$	7000$	5700$	....
Sushi & Go	19000$	9900$	15000$	8500$	

Fig. 4.7 Resulting data bundle of applying pivot to the one of Figure 4.6.

an aggregation function associated to each pivoted property, so that each aggregation function accepts as input a tuple of values of the associated pivoted property type, and produces as output a single value of an arbitrary type.

We explain these concepts with the help of the *FinancialResults* data bundle of Table 4.6. The objects, or class instances of this data bundle are simple objects without nested objects, so they can be more easily represented as table rows rather than as JSON objects. Using this example, we might pivot the *FinancialResults* bundle of Table 4.6 using as static properties just the *business name*, as pivoting columns *year*

and *month*, and, as pivoted columns, *income* and *outcome* (represented as *in* and *out* in the picture for space reasons). As aggregation function for each pivoted property, we might use $sum : Tuple\{\mathbb{R}\} \rightarrow \mathbb{R}$.

Using these inputs, the pivot operation produces as output a new class A', which has as properties the static properties of A plus a set of new properties. To calculate these new properties, we extract first all distinct tuples corresponding to the pivoting properties of A. Then, we derive a string representation of each one of these tuples using the *stringify* auxiliar function. In our example, the distinct tuples for the pivoted properties *year* and *month* would be *((2018, Jan), (2018, Feb), (2018, Mar), ...)*. After being *stringified*, these tuples would produce the set { "2018_Jan", "2018_Feb", ... }. We refer to this set as the *prefixes* set.

Then, for each pivoted property and for each element in the prefixes set, we create a new property. The name of each new property is formed by concatenating the prefix element and the property name being processed. As type, the property has the type of its corresponding aggregation function, and, as cardinality, the cardinality of the original pivoted property. In the example, we would generate as new properties *{(2018_Jan_in,* $\mathbb{R}$*, (1,1)), (2018_Feb_out,* $\mathbb{R}$*, (1,1)), ...}*. As it can be seen, the number of properties of the output class can be much larger than the number of properties of the input class. In our example, twenty-four new properties would be created per year contained in the instances of the input data bundle.

After defining the structure of the output class, their instances are computed. For this purpose, all distinct tuples corresponding to the static properties of A are calculated. In our example, just the *(Pete's Pizza)* and *(Sushi & Go)* tuples would be retrieved.

To complete each one of these tuples with values for the newly created properties, the following non-trivial process is executed. Let p be a newly created property of a tuple a. For instance, let p be the *2018_Jan_in* property of *(Pete's Pizza)*. To calculate this value, we search for all instances of A whose static properties match with a, and whose pivoting properties match with the values corresponding to p. In our case, we search for instances of A having as values *(Pete's Pizza, 2018, Jan)* for the *name*, *year* and month properties, respectively. We would get four instances in our case, one per week in a month. Next, we extract the value of each one of these instances for the pivoted property associated to p. This is, we would extract the value for the *Income* property of each one of these four instances. Finally, all these values are reduced by applying the corresponding aggregation function. The result is used as value for the newly created property. In our case, we would calculate the sum of the

four extracted values, and this value would be placed in the *2018_Jan_in* property of the *(Pete's Pizza)* tuple.

As the reader can notice, while we mentioned that the number of properties of A' can be considerably higher when compared with A, the number of instances of A' would be lower. This happens because the pivot operation is compressing replicated instances in a single one, spreading the information of each replica over the new properties.

It could be the case that, for some of these new properties, we do not have any values to aggregate for certain instances. For example, if the *Pete's Pizza* business was closed in August 2018, we would not have income values for any week of the generated property *2018_August_in*. If this happens, then the value for such properties is left undefined, i.e., *2018_August_in = ()* for the *Pete's Pizza* business.

One special pivot case happens when, after defining the *static* and *pivoting* property sets, there are no properties left to be pivoted. For instance, we might want to pivot a *BusinessCategories (businessName, category)* class, whose instances store business-category pairs, meaning that such a business belongs to the related category. For this operation, we would select $\{businessName\}$ as static properties, and $\{category\}$ as pivoting properties, while the pivoted properties set would be empty. The pivot operation would take place as follows: a boolean property would be generated for each *category* value, with the same name as the category. The value of each category property would be *true* for those businesses related with that category in the *BusinessCategories* data bundle, and undefined for the not related.

Finally, it is worth to mention that, if we wanted to extend the Algorithms 2, 3 and 4 to work with properties with cardinality higher than 1, we would need only to modify the profiles of the *stringify* and *aggregate* functions so that they work with tuples of tuples of values instead of just tuples of values.

Next sections describe how the flattening operator can be defined with the help of the *join* and *pivot* operations.

4.4.3 Trivial Case: Single Class, Single-Value Attributes

The trivial input for a flattening operation is a class that just contains single-value attributes. Figure 4.8 (left) provides an example of such a class. In this example, we want to create a table, or dataset, for the *Review* class, which contains the *r_id*, *stars* and *text* attributes. The flattening process for this case, which is described in Algorithm 5, is straightforward. The operation accepts as input a data bundle, where the class only contains the definition of single-value attributes, and produces as output a table. To define the table header, for each property in the class, a column is created

Review
r_id : string stars : double text : string

r_id	stars	text
R1	4.5	We were recommended this by ...
...	...	...
R2	3.7	The first impression was not ...

Fig. 4.8 Left: Main class (Review) to be transformed; right: The resulting table.

Algorithm 5: Flatten - Trivial Case.

Input: A data bundle $A = (A_{class}, A_{data})$
Output: A table $T = (T_{header}, T_{data})$
1 Let $A_{class} = (name_A, Prop_A)$, $T_{header} \leftarrow ()$;
2 **foreach** $prop \in Prop_A$ **do**
3 | Let $prop = (name, type, cardinality)$;
4 | $T_{header} \leftarrow T_{header} \bullet (name)$;
5 **end**
6 $T_{data} \leftarrow A_{data}$;

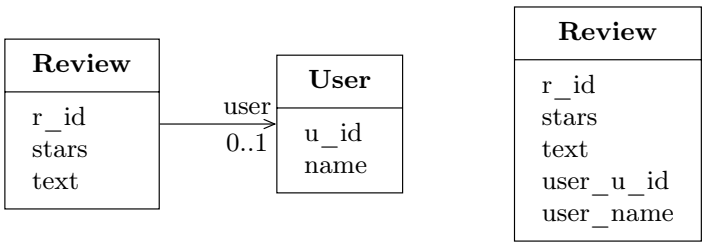


Fig. 4.9 One-bounded association.

with the same name. Then, each instance of the class is copied as a row in the output table. All the other patterns that can be found in an object-oriented model are reduced to this base case.

In our example, we would construct a table with columns *r_id*, *stars* and *text*. Then, each *Review* instance would be added as a row to this table.

4.4.4 Single-Bounded Reference

The second pattern corresponds to the case of a class A containing single-valued attributes and one reference ref with upper bound 1 to a class B, which contains single-valued attributes and no references. This case is illustrated in Figure 4.9 (left), where we wish to flatten the *Review* class, now containing a single reference *user* to a *User* class.

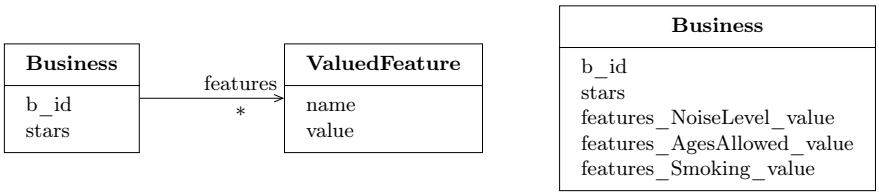


Fig. 4.10 Reduction of an unbounded reference.

To reduce this pattern to the base case, we need to combine both classes in a single class and remove the reference ref. This is achieved by joining the class A with the class B through the reference ref. After performing the join, the resulting class A' contains the same instances as A, since each instance of A is combined with at most one instance of B; and all attributes of A and B, but the reference ref would have been removed. So, after the join, the pattern is reduced to the trivial case. In summary, the flattening of this pattern can be formalised as follows:

$$flatten(A) = flatten(join(A, ref))$$

Figure 4.9 (right) shows the result of this join of the *Review* class with the *User* class through the *user* reference. As it can be seen, this class can now be flattened using the trivial case strategy.

4.4.5 Unbounded Reference

In this new pattern, we have, as before, a class A containing single-valued attributes and one reference ref to a class B, which contains single-valued attributes and no references, but now ref is unbounded. Figure 4.10 illustrates this case with an example, where we want to flatten the *Business* class, which has an unbounded reference to a *ValuedFeature* class.

In this pattern, unlike in the previous one, each instance of A can relate to many instances of B. Consequently, if we simply join classes A and B, we would have a class A' with all attributes of A and B, but the data corresponding to each instance of A would be replicated in several instances of A', such as in Figure 4.5 (b). Therefore, a join operation would not be enough in this case, as it would not satisfy all the constraints imposed by data mining algorithms.

To solve this problem, we can compact the output of the join operation using a pivot. Since we want to have one row per instance of A, we have to use all properties

of A as static properties for the pivot operation. This ensures that the output of the pivot operation has as many instances as A has. Thus, using some of the attributes of B as *pivoting properties* and the remaining of then as *pivoted properties*, the pattern would be reduced to the trivial case.

The problem is now precisely how to decide which attributes of B would become *pivoting properties* and which ones would be *pivoted properties*. Since we want to preserve all information of each instance of B related to each instance of A, each instance of B should be spread over a set of attributes reserved for that instance. For instance, in our example, the information of each *ValuedFeature* associated to one business should be placed in a set of attributes created for that *ValuedFeature*.

A solution to achieve this is to use as *pivoting properties* a set of attributes of B that uniquely identify each instance of B inside the collection represented by the reference being processed. In our example, we want to find an attribute that can identify each *ValuedFeature* inside the *features* collection of a *Business*. In this case, the *name* attribute can play this role. Finding these attributes requires knowing the semantics behind each attribute. Since the flatten operator is only able to analyse the syntactic structure of a pattern, it cannot compute these attributes by itself. Consequently, the user must provide these attributes as an input for the flattening operator in this case. These user-specified attributes of B are used as *pivoting properties* and the remaining ones as *pivoted properties*.

It must be taken into account that we are pivoting the result of the join of A and B, this is, A'. In A', the name of the properties of B are prefixed with the name of the reference used for the joins, so that name collisions in A' are avoided. Consequently, when performing the pivot, the names of the *pivoting* and *pivoted properties*, coming from B, need to be updated to match with those in A'. With all these considerations, the flattening of this pattern can be formalised as follows:

$$
\begin{aligned}
&flatten(A, \{(ref, Pivoting)\}) = \\
&\qquad flatten(pivot(join(A, ref), \\
&\qquad\qquad Prop_A, \\
&\qquad\qquad updateNames(Pivoting), \\
&\qquad\qquad updateNames(Prop_B - Pivoting)))
\end{aligned}
$$

For our example, if we use the *name* property of *ValuedFeatrure* as pivoting property, the flatten operator would firstly join the *Business* and *ValuedFeature* data bundles through the *features* reference. This resulting data bundle is pivoted, us-

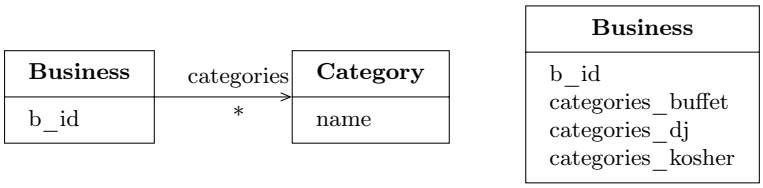


Fig. 4.11 Special unbounded reduction where no value attributes are present (the category name is used as pivoting attribute).

```
[{name : "Pete's Pizza", stars : 4.5,
  categories : ["buffet", "dj"]},
 {name : "Sushi & Go", stars : 3.8
  categories : ["buffet", "kosher"]},
 {name : "Wine Heaven", stars : 4.0,
  categories : ["buffet", "dj","kosher"]}]
```

Fig. 4.12 Businesses with their categories represented as a multivalued attribute.

ing $\{b\_id, stars\}$ as static properties; $\{features\_name\}$ as pivoting properties; and $\{features\_value\}$ as pivoted properties. Figure 4.10 (right) shows a potential output class for this process. In this case, we have considered there are three possible *names* for a *ValuedFeature*: *NoiseLevel*, *AgesAllowed* and *Smoking*. Therefore, three new attributes are created to host the values associated to these features.

In Figure 4.11, the example that is commented in the special pivot case at the end of Section 4.4.2 is depicted. No pivoted attributes are present in this case ($Pivoted = \emptyset$), as the Category class only has one attribute, *name*, which is used as *pivoting* property. Therefore, the generated attributes indicate whether each business's *categories* reference points to any of the possible categories (*buffet*, *dj* or *kosher* in the example). The flatten operation would be *flatten(Business, (categories, {name})*.

4.4.6 Multi-Valued Attributes

Another potential input pattern might be a single class, with no references, but with one multivalued attribute. A *multivalued attribute* is an attribute, i.e., a property with a primitive type, but with an upper bound greater than one. For instance, in the Yelp conceptual model, the *Categories* class only has one string attribute, *name*. In addition, this class is only related to the *Business* class. Therefore, these categories could also be registered in the model by including a multivalued attribute, of type String, into the *Business* class. Figure 4.12 depicts this example with a JSON file where the categories of each business are registered through an array of strings.

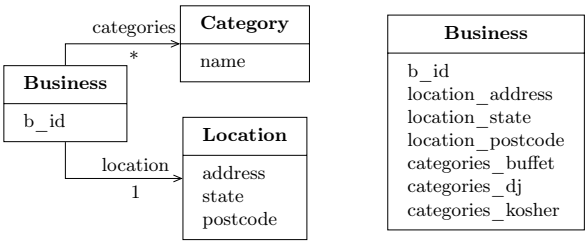


Fig. 4.13 Multiple references reduction.

Since multivalued attributes can be viewed as a degenerated case of unbounded references, we can follow a similar strategy and rely on the pivot operation. For this operation, we would use as *pivoting* values the set of values of the reduced attribute, and as *pivoted* value the mere presence of each individual value in the multivalued attribute of a class instance, just as in the special pivot case commented at the end of Section 4.4.2.

In the Yelp example, if *categories* were a multivalued attribute of *Business* as described in the previous paragraph, a pivot operation could be invoked by using the different category names as *pivoting* values, and the presence of such category names in the multivalued attribute of each business instance would be used as *pivoted* values for each business. This would be conceptually equivalent to firstly transforming the example of Figure 4.12 into the case of Figure 4.11 and then reducing it.

4.4.7 Multiple Reductions

Now that we have seen the different ways to flatten a reference, we can start with those where several references are reduced. In Figure 4.13, we are reducing two references from the *Business* class, namely, *location* and *categories*.

When reducing multiple references from the main class, the strategy is to reduce each reference individually, using the previously described patterns. As each reduction generates its own set of new properties, these properties do not collide and, therefore, it does not matter the order in which they are executed. It is worth to hightlight that, as it was specified in the unbounded references section, the reduction of an unbounded reference requires the specification of the pivoting properties. Consequently, when multiple unbounded references are reduced, we need to specify, for each unbounded reference, which properties of the referenced class will be used as pivoting properties. Therefore, the flatten operator needs in this case as extra argument the collection of (unbounded reference, pivoting attributes) pairs that provide such information.

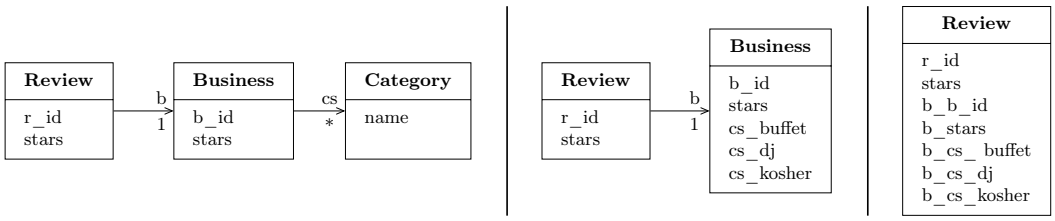


Fig. 4.14 A two-step reduction of multilevel references.

Based on this description, the command to accomplish the reduction of Figure 4.13 would be as follows:

$$flatten(Business, \{(categories, \{name\})\})$$

When executing that command, the flatten operator would reduce the *location* reference using the single-bounded pattern of Section 4.4.4; and then, the *categories* reference with the unbounded pattern of Section 4.4.5. For this second reduction, the *pivoting* properties set is required, which is passed as a parameter of the operator. As commented, the order of these reductions can be swapped.

4.4.8 Multi-Level Reductions

Until now, we have considered that a referenced class does not have its own references. In this subsection, we extend the previous patterns by allowing referenced classes to also reference other classes, which might also have references again, and so on. Figure 4.14 (left) shows an example of this pattern, where we wish to reduce the information of three connected classes: *Review*, which plays the role of main class, *Business* and *Category*. Reference names have been abbreviated for space reasons.

The strategy to reduce a chain of references to a single class is to reduce step by step each class in the chain to a single class, going from tail to head, until compacting the full chain into the main class. In the case of Figure 4.14, the *cs* reference is reduced first, and the *Category* class gets merged into *Business* class by means of a join plus a pivot, according to the pattern for unbounded references. The result of this step is shown in Figure 4.14 (middle). After this first step, we would reduce the *b* reference using the pattern for single-bounded references, obtaining the result of Figure 4.14 (right).

So, the command to achieve this reduction is:

$$flatten(Review, \{(cs, \{name\})\}) \tag{4.1}$$

As it can be observed, similarly to the case of multiple references, for each unbounded reference to be reduced, we need to specify which properties of the referenced class should be used as pivoting properties, such as for the *Categories* reference.

4.4.9 Inheritance

In all previous patterns we ignored that object-oriented models can contain inheritance relationships. This section describes how inheritances are handled by the flattening operator. Before that, we show how we extended the formalisation provided in Section 4.4.1 to support inheritance relationships.

Inheritance Formalization

We extended the definition of *class* and *instance*, so that a class can have *superclasses*. For this purpose, we also introduced two new functions, *properties* and *isKindOf*.

Definition 4.4.18. *A **class** C is a tuple $(n, Properties, Superclasses) \in String \times Set\{Property\} \times Set\{Class\}$, where n is the name of the class, $Prop$ is the set of properties of the class, and $Superclasses$ is a set of classes that are superclasses of this class.*

Definition 4.4.19. ***properties*** $: Class \rightarrow Set\{Property\}$ *is a function that, given a class $C = (n, Prop, Super)$, where $Super = \{SC_1, \cdots, SC_n\}$, then $properties(C) = Prop \cup \bigcup_{i=1}^{n} properties(SC_i)$.*

Definition 4.4.20. ***isKindOf*** $: Type \times Type \rightarrow Boolean$ *is a function that, given two types t_1 and t_2,*

1. *If $t_1 = t_2$, then $isKindOf(t_1, t_2) = true$.*
2. *If $t_1, t_2 \in Class$, $t_1 = (name, Prop, SC)$ and $t_2 \in SC$, then $isKindOf(t_1, t_2) = true$.*
3. *Otherwise, $isKindOf(t_1, t_2) = false$.*

For the sake of simplicity, we consider that our input models are free of name conflicts due to the presence of multiple inheritances. This is, a class cannot inherit a

property with the same name from more than one superclass. If these conflicts are detected in the conceptual model, they are reported to the designer, who must fix this model by modifying a property name.

Definition 4.4.21. *Given a class C, with $properties(C) = \{p_1, \cdots, p_n\}$, and $p_i = (n_i, t_i, (c_{min_i}, c_{max_i}))$, an **instance** is a tuple $(values_1, \cdots, values_n)$ where $values_i$ is a tuple with $c_{min_i} \leq |values_i| \leq c_{max_i}$, $isKindOf(typeOf(v), t_i)$, $\forall v \in values_i$.*

Definition 4.4.22. ***propValues*** *: $Instance \times Property \rightarrow Tuple\{LiteralValue\} \cup Tuple\{Instance\}$ is a function that, given an instance $ins = (values_1, \cdots, values_n)$ of a class C with $properties(C) = \{p_1, \cdots, p_n\}$, $propValues(ins, p_i) = values_i$.*

General Strategy for Inheritance Reduction

When applying a reduction pattern, if a class in the pattern is part of an inheritance hierarchy, we first compact this hierarchy into the mentioned class, and then the pattern can be reduced as usual. In the following, we denote the class over which the inheritance hierarchy is compacted as the *reduction class*.

This reduction class can be at any level of an inheritance hierarchy: either the root class, a leaf, or an intermediate class. So, as it can be noticed, when compacting an inheritance hierarchy into a reduction class, we are including in that class all properties coming from their super and subclasses. Including superclasses properties is natural, since these properties are, by definition, properties of the reduction class too. Including subclass properties is also considered in our case, because these properties might be of interest in some data analysis.

The way of including subclasses properties is borrowed from the *Single Table Pattern* used by Object-Relational Mappers (Fowler [60]). Algorithm 6 details this operation. This pattern adds any attribute coming from a subclass to the reduction class. Since two subclasses might define a property with the same name, the prefix *sub_<ClassName>_* is added to the name of each subclass property to avoid name collisions. Finally, a new property *type* is added to the reduction class. This property is used to know the concrete class of the inheritance hierarchy to which each instance belongs.

When compacting a hierarchy, we consider that all classes in the hierarchy are classes made up of just single-bounded attributes. If not, those classes that do not fulfill this constraint should be reduced to a single class using the previously specified patterns, and then the hierarchy would be compacted.

Algorithm 6: Inheritance Reduction - General Case

Input: A data bundle $A = (A_{class}, A_{data}), A_{class} = (name_A, Prop_A, SC_A)$
Output: A data bundle $A' = (A'_{class}, A'_{data}), A'_{class} = (name_{A'}, Prop_{A'}, \emptyset)$
// Construct A′ class
1 $name_{A'} = stringify(name_A, \text{“ ′ ”})$;
2 $Prop_{A'} = Prop_A$;
3 Let $Subclasses_A = \{C \mid C \in Class, C \neq A_{class}, isKindOf(C, A)\}$;
4 **foreach** $C = (name_C, Prop_C, SC_C) \in Subclasses_A$ **do**
5 **foreach** $p = (name_p, t, (c_{min}, c_{max})) \in Prop_C$ **do**
6 $newP \leftarrow (stringify(\text{“}sub\_\text{”}, name_C, \text{“}\_\text{”}, name_p), t, (0, c_{max}))$;
7 $Prop_{A'} \leftarrow Prop_{A'} \cup \{newP\}$
8 **end**
9 **end**
10 $Prop_{A'} \leftarrow Prop_{A'} \cup \{(\text{“}type\text{”}, String, (1, 1))\}$;
// Construct A′ data
11 **foreach** $ins \in A_{data}$ **do**
12 ins is an instance of class $C(name_C, Prop_C, SC_C)$, $isKindOf(C, A_{class})$;
13 $newIns \leftarrow ins$;
14 **foreach** $prop \in A'_{class}$ **do**
15 **case** $prop \in properties(C)$ **do**
16 $propValues(newIns, prop) = propValues(ins, prop)$;
17 **case** $prop = propType$ **do**
18 $propValues(newIns, propType) = name_C$;
19 **otherwise do**
20 $propValues(newIns, prop) = ()$;
21 **end**
22 **end**
23 **end**

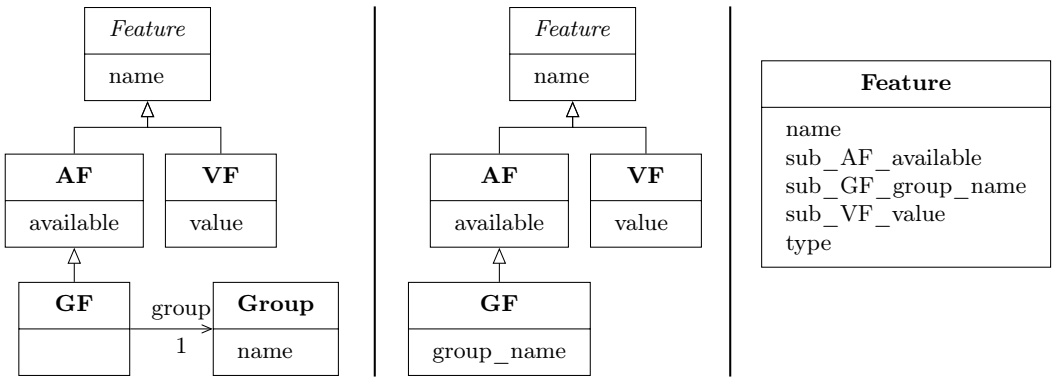


Fig. 4.15 Inheritance Reduction - General Case

This process is illustrated with the example of Figure 4.15 (left), where we the *Feature* class is the main class, and consequently, the reduction class of that hierarchy. To compact this hierarchy, since the *GF* class has a single reference to *Group*, this reference is reduced first with the pattern of Section 4.4.4 (Figure 4.15 (middle)). Then, all the *Feature* subclasses' properties are merged into this class with their corresponding prefixes. Then, the new *type* property is added, generating the result shown in Figure 4.15 (right).

One problem associated to the *Single Table Pattern*, and, consequently, to our strategy for inheritance reduction, is that those properties coming from a specific subclass are not defined in those instances belonging to other subclasses. For example, in Figure 4.15, instances of *VF* would not have any value for the properties *sub_AF_available* or *sub_AF_group_name*. This might reduce the quality of the analysis results, since some data mining algorithms might have problems dealing with null values. Nevertheless, this is the price we must pay if we want to consider subclass-level information. However, there is a particular case where this problem can be mitigated. This case is described in the following section.

Special Case: Reduction Class in Unbounded Reference

In the specific case where the reduction class is part of an unbounded reference, the null values problem previously described gets worse, since the reduction of unbounded references generates several sets of properties, one per value of the pivoting properties (see Section 4.4.5).

Figure 4.16 is used to illustrate this case. Each *Business* has an unbounded reference to the *Feature* class, which is the root of the hierarchy that we reduced previously (see Figure 4.15).

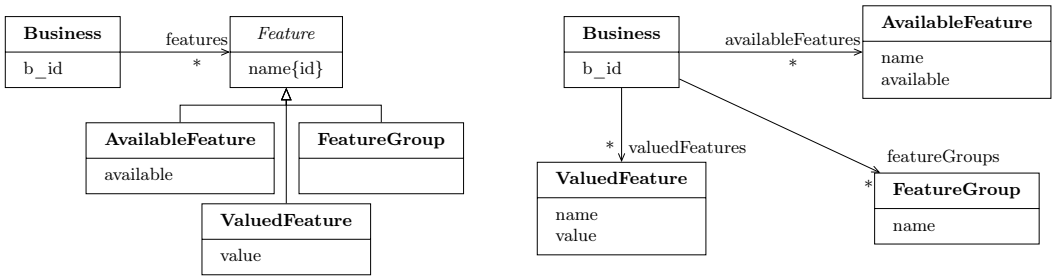


Fig. 4.16 Left: *Business* and its reference to the *Feature* inheritance; right: type division of the *features* reference performed in the special subclass reduction.

Applying the general case reduction (Algorithm 6), the properties of *Feature* would be augmented with the set *{sub_AF_available, sub_VF_value, sub_GF_group_name, type}*. This is, we move from one property in *Feature* to five. When reducing the *features* unbounded reference, these properties are replicated for each one of its associated *pivoting* values. Assuming we use *name* as pivoting property, all this set of properties would be used for each existing feature.

Nevertheless, in this case, it might be not necessary to include all subclasses properties when pivoting a specific instance of the *Feature* class. For example, let us focus in a specific feature instance whose *name* is *AgesAllowed*. This feature would be a *ValuedFeature* that can take as value an indication of an age restriction for a business (e.g. "18plus" or "21plus"). Therefore, when pivoting such an instance, only two properties out of the five contained in the compacted *Feature* class are needed. This is, for the group of properties associated to the *AgesAllowed* pivoting value, just the *name* and *value* properties are required, so the other properties can be removed. Any extra property included in this set would be undefined, since *AgesAllowed* is always an instance of *ValuedFeature*, and not of any other subclass.

Taking this into account, we can apply the following strategy to reduce the number of null values: one new unbounded reference is generated for each one of the subclasses of the reduction class. These references would then be populated with the instances from the original unbounded reference belonging to each concrete type. Therefore, references pointing to the subclasses, when reduced, would replicate just the properties of each concrete class, decreasing the number of null values in the resulting dataset. This process is more precisely specified in Algorithm 7.

In Figure 4.16, the *features* reference would be split into one reference for each class in the hierarchy. For instance, for the *ValuedFeature* class, a *features_ValuedFeature* reference is generated. The *Feature* class does not get a reference because it is an abstract class, so no instances can be generated for this type. References are populated

Algorithm 7: Unbounded reference splitting method.

Input: A data bundle $A = (A_{class}, A_{data}), A_{class} = (name_A, Prop_A, SC_A)$
Input: A reference $ref = (name_{ref}, B, c), B = (name_B, Prop_B, SC_B)$
Output: An updated data bundle A

Let $Subclasses_B = \{C \mid C \in Class, C \neq B, isKindOf(C, B)\}$;
$NewRefs \leftarrow \emptyset$;
foreach $C = (name_C, Prop_C, SC_C) \in Subclasses_B \cup \{B\}$ **do**
    $newRef \leftarrow (stringify(name_{ref}, \text{“\_”}, name_C), C, (0, *))$;
    $NewRefs \leftarrow NewRefs \cup \{newRef\}$;
end
$Prop_A \leftarrow Prop_A \cup NewRefs$;
foreach $ins \in A_{data}$ **do**
    $RefValues \leftarrow propValues(ins, ref)$;
    foreach $newRef = (name_{newRef}, C_{newRef}, c) \in NewRefs$ **do**
        $propValues(ins, newRef) \leftarrow (values \mid$
          $value \in RefValues \wedge typeOf(value) = C_{newRef}$;
    end
end
$Prop_A = Prop_A - \{ref\}$;

with the instances of their type present in the *features* reference: *ValuedFeatures* originally contained in *features* would be moved to the *features_ValuedFeature* reference, *AvailableFeatures* to *features_AvailableFeature*, and so on. After this process, each one of the new references can be reduced into the *Business* class as explained in Section 4.4.5.

4.5 Lavoisier: Dataset Extraction Language

Once the *flattening operator* was fully specified, we built a language, based on that operator, to perform dataset extractions from conceptual models. We named this language *Lavoisier*. Listing 4.2 shows the minimal example of a Lavoisier snippet.

Listing 4.2 Lavoisier's simplest dataset specification.

```
1 dataset yelp_reviews {
2   mainclass Review
3 }
```

In Lavoisier, extractions are expressed by defining *dataset specifications*, which are expressed with the *dataset* keyword followed by a *name* (for instance, *yelp_reviews* in Listing 4.2), and a *body* block surrounded by braces (lines 1-3). A dataset specification

must always declare a *main class* (line 2), which is the class whose instances would be placed in each row of the output dataset. If we do not include any further information, a flattening process is applied to this main class, including all its attributes and excluding all its references. This is a pragmatic convention, but not enough for most cases, so Lavoisier provides some options to modify this default behaviour. These options are commented in the following.

4.5.1 Properties Selection

If we do not want to include all attributes of the main class, we can select a subset of them by specifying a list of attributes between brackets. This list appears after the main class specification. In Listing 4.3, line 2 only the *r_id* and *stars* attributes of a *Review* are selected to be included in the output dataset.

Listing 4.3 Reviews data with some attribute and reference selections.

```
1 dataset yelp_reviews {
2   mainclass Review [r_id, stars]
3   include user
4   include business {
5     include location[address, postalCode]
6     include categories by name
7   }
8 }
```

The selection of references is different from that of attributes because of two main reasons: (1) referenced types can have nested attributes and references that we may need to manage; and (2) references might be unbounded, which introduces the need to specify a set of attributes to play the role of *pivoting* attributes in the reduction (see Section 4.4.5). Therefore, the treating of references might require extra configuration as compared with the syntax for selecting attributes.

Single-bounded references can be incorporated to a dataset through the *include* keyword. For example, in Listing 4.3, line 3, the *user* reference is included in the dataset through the *include user* statement. If no extra information is given, all attributes of the included class (e.g. *User* in this case) would be flattened into the *main class*, i.e., *u_id* and *name*.

In line 4 of Listing 4.3, another single-bounded reference is included in the dataset specification, *business* in this case. If we want to customise which elements of the reduced reference are included, we can do so by adding a block to the *include* construct, such as the one used in the case of *business* (lines 4-7). Inside this block, we can use

the same modifiers as in the main class to include more references, and we can keep including references up to the nesting level that is required for each concrete dataset specification.

Inside the *business* inclusion block, first the *location* reference is included (line 5). From this reference, just the *address* and *postal code* attributes are selected to be included in the output dataset. This selection was performed by indicating these attributes between brackets after the reference name, just as we did with the attributes of the main class at line 2 of the listing. Then, *categories* is included, which is an unbounded reference (line 6). As an additional parameter for unbounded references, we need to specify the set of attributes from the included reference that should be used as *pivoting* attributes. This is done with the *by* keyword after the name of the unbounded reference. For the example, the *name* of the categories is used as pivoting attribute.

4.5.2 Inheritance Management

As we saw in Section 4.4.9, inheritances may appear in the main class of a flattening operation, or in one of the reduced references. In both cases, if no extra information is provided, Lavoisier by default reduces inheritances by including all attributes of the super and subtypes of the reduction class, either by the properties collapse reduction or, if reducing an inheritance present in an unbounded reference, by the reference splitting method (see Sections 4.4.9 and 23, respectively).

On the other hand, as users of Lavoisier might just be interested in some of the types of an inheritance, or they could want to include any of the references of a concrete type, the language includes support for customising this reduction process. First, attributes and references contained in the superclasses of the reduction class can be included in the dataset with no special syntax, i.e., by making use of the attribute selection and reference inclusion constructs that we introduced in the previous section. For the selection of properties from subclasses, a new pair of constructs was introduced: *as* and *only as*.

Listing 4.4 Inclusion of *GroupedFeature's group* reference in the inheritance reduction.

```
1 dataset yelp_businesses {
2   mainclass Business[name, stars]
3   include features by name {
4     as GroupedFeature { include group }
5   }
6 }
```

Listing 4.4 shows a dataset specification where business data is being extracted. Among the selected business' properties, the *features* unbounded reference is included (lines 3-5). As we know, features are represented through an inheritance hierarchy. Therefore, Lavoisier would apply the reference splitting inheritance reduction (see Section 23) including only attributes from the involved classes: *Feature*, *AvailableFeature*, *ValuedFeature* and *GroupedFeature*. For the last of these classes, a very important property is present in a reference: the *group* of the property. By default, Lavoisier would omit this reference when reducing the inheritance, as this language only processes attributes. If we want to also include certain references, we can do so with the *as* construct. This construct allows selecting the concrete properties of a type that are to be flattened. For instance, to include the *group* feature, we have to specify the *as* keyword followed by the name of the type to customise (i.e. *GroupedFeature* in this case) and by a block, where we can make use of inclusion mechanisms, such as the *include* construct that is used to select the *group* reference in line 4.

Listing 4.5 Selection of only two subclasses from the inheritance for the reduction.

```
1 dataset yelp_businesses {
2   mainclass Business[name, stars]
3   include features by name {
4     only as AvailableFeature {}
5     only as ValuedFeature {}
6   }
7 }
```

The previous type customisation only affects the type being modified, as any other type in the inheritance would be processed through the default behaviour (i.e. all attributes processed, no references). Another configuration we might want to do when reducing an inheritance is to only reduce some concrete types of such an inheritance, instead of all of them. We can do so by using the *as* construct, but preceding it with the *only* keyword. In Listing 4.5 shows a dataset specification where two *only as* statements are used in the reduction of the *features* reference (lines 4-5). These statements refer to the *AvailableFeature* and *ValuedFeature* classes, which means that *only* these types of the inheritance would be flattened in the reduction. Any other type of the inheritance would be omitted.

4.5.3 Instances Filtering

It might also the case that we are not interested in including al instances of a class in a particular dataset. For example, in the Yelp case study, we might want to limit an

analysis to the reviews of businesses of a specific city. This can be achieved in Lavoisier by means of filters. A *filter* is specified after a class or reference name (or the attribute selection list, if present), using the *where* keyword. A filter contains a predicate that is evaluated for each instance. If the predicate evaluates to *true*, then the instance is processed; otherwise it is discarded.

Listing 4.6 Lavoisier's simplest dataset specification.

```
1 dataset yelp_reviews {
2   mainclass Review [r_id, stars]
3     where business.location.city = "Edinburgh"
4   ...
5 }
```

Listing 4.6 shows a *Lavoisier* snippet using a filter, where we are interested in selecting only those reviews from the city of Edinburgh. This restriction is specified with the *where* keyword followed by a boolean equality which compares the city of a business to the string "Edinburgh" (line 3).

4.5.4 Derived Values

It is worth to comment that, when applying the pattern for reducing unbounded references (see Section 4.4.5) between a class A and a class B, it could happen that certain instances of A are not related to certain instances of B. For example, if the set of features defined by each business is not somewhat homogeneous, this means, each business defines a set of features that others seldom define, the execution of Listing 4.4 would genearte as output a dataset with a non-negligible amount of undefined feature values. Undefined values often hamper the quality of results of data mining algorithms. So, we must take care of not flattening unbounded references that can generate large amounts of undefined values.

In an extreme case, the set of instances contained in the unbounded reference of an instance of the main class could not have any relation with the sets of other instances. This phenomenon happens with the *reviews* that each business instance has: this set is personal for each business, and disjoint of the reviews that other businesses have received. Therefore, references like *reviews* should not be flatenned, as the generated dataset would have poor quality. Instead, these references should be processed by calculating some statistical values that summarise them. For instance, we might calculate the number of reviews received by each business, or the number of these reviews that were positive (e.g. that gave 4 or more stars).

Listing 4.7 Lavoisier's *calculate* construct to perform aggregations.

```
1 dataset yelp_businesses {
2   mainclass Business[name, stars]
3   calculate numReviews
4     as count(reviews)
5   calculate numPositiveReviews
6     as count(reviews) where stars >= 4
7 }
```

To support these derived values, Lavoisier provides users with the *calculate* primitive. Listing4.7 shows an example where this primitive is used. The *numReviews* column is calculated by applying the *count* function over the *reviews* reference of a business (lines 3-4). The *numPositiveReviews* is obtained through a similar calculation, but this time only those reviews with 4 stars or more are counted (lines 5-6). This selection is performed with a *where* clause, just as instances were filtered in Section 4.5.3.

The main limitation of the *calculate* primitive is that it only supports spreadsheet-like aggregation functions, such as *count*, *avg*, *min* or *max*. Although these functions can be combined with the instances filtering mechanism to provide an acceptable set of basic calculations, they might not be enough for obtaining more complex derived values, such as the *ratio of positive vs. negative reviews during the last summer*. If these complex values are used somehow as KPIs (*Key Performance Indicators*) in the domain being analysed, they should be included in the conceptual domain model, not being responsibility of Lavoisier to calculate them. However, it might be case that, in the light of new facts identified when analysing a data bundle, new KPIs arise. In this case, it would be interesting that Lavoisier provides facilities to calculate these values. The main problem would be that, to include support for calculating these values, a small but very-close-to-programming language might be required, and the skills to manipulate such a language might be out of the scope of those average decision makers for which *Lavoisier* was designed.

4.5.5 Implementation

Lavoisier has been implemented with the Xtext framework (Eysholdt and Behrens [55]), and the conceptual models queried by this language are represented in the *Ecore* format (Steinberg et al. [152]). Through the usage of Xtext, we obtain a very capable editor, with easy inclusion of terms proposal and validation into the language. The user gets assisted through the dataset specification process, which is validated against

the conceptual model to ensure correctness and to provide useful suggestions. The implementation of Lavoisier is freely available in an external repository[3].

When executing a Lavoisier script, a *CSV (Comma-Separated Values)* file is generated for each dataset specification. First, the instances of the main class to be included in this the output dataset file are gathered, taking into account the specified filters. Then, the attribute and reference inclusions constructs are processed to determine the different sets of columns that must be generated, in the same order that they were defined. Lastly, these columns are calculated for each gathered instance of the mainclass, and placed as rows in the output dataset.

Notice that, although Lavoisier snippets seem small and simple, a considerable number of entities from the model intervene in some of the presented dataset specifications. For instance, 5 entities are involved in the specification of Listing 4.3. This example might suggest the power offered by Lavoisier's apparently simple constructs, in part thanks to the defined flattening patterns that this language uses internally. To offer a more objective study of the benefits of Lavoisier, in the next section we compare this language with some state-of-the-art technologies for data flattening.

4.6 Evaluation

This section evaluates whether Lavoisier is an effective language for the creation of datasets from a high-level perspective, without the need of having to perform low-level data transformation operations like *joins* or *pivots*.

In this evaluation, we firstly analysed whether Lavoisier is expressive enough to create datasets for a wide range of contexts. Secondly, we studied in detail the advantages provided by Lavoisier when compared with state-of the-art languages used for defining datasets. More specifically, we compared *Lavoisier* against the SQL (Beighley [18]) language and the Pandas (McKinney [114]) data transformation library. These technologies have been selected as representatives of the main groups discussed in Section 4.3.

The comparison was carried out by analysing *conciseness* and *specification complexity* of different dataset extraction scripts developed using Lavoisier, SQL and Pandas. These scripts were devised to cover a wide range of dataset creation scenarios. By analysing *conciseness*, or script size, we expected to measure how much boilerplate code can be saved due to the use of Lavoisier; whereas by comparing *specification*

[3] https://github.com/alfonsodelavega/lavoisier

Table 4.1 Lavoisier support for dataset creation tasks.

Category	Subtasks	Supported
Structural	Single table	Yes
	Single-bounded reference	Yes
	Unbounded reference	Yes
	Multi-valued attributes	Yes
	Multiple reductions	Yes
	Multi-level reductions	Yes
	Inheritance	Yes
Customisation	Attribute filter	Yes
	Instance filter	Simple
	Subclass selection	Yes
Calculation	Aggregations	Simple
	Derived features	Pending

complexity, we aim to know how many operations and parameters need to be specified when using each one of the compared technologies.

4.6.1 Expressiveness

To assess Lavoisier's expressiveness, we checked whether and how this language supports those tasks that are typically carried out when producing a tabular dataset during a data mining process (see Figure 4.2, Step 5). These typical tasks were grouped into three categories: *Structural*, *Customisation*, and *Calculation*. The *Structural* category contains those tasks where certain pattern of the conceptual model is reduced to a simpler form, e.g., the reduction of a class with an unbounded reference to a single class. The *Customisation* category covers all operations related to the personalisation of the extracted data, for instance, by filtering out certain instances of a particular class. Finally, the *Calculation* category refers to those tasks related to the definition of new features based on aggregation functions or arithmetical operations.

We decomposed each one of these categories into several subtasks, and, for each subtask, we analysed how well Lavoisier addresses it. Table 4.1 shows the results of this decomposition and analysis process.

The structural category was decomposed into seven different subtasks, which match with the transformation patterns identified for the *flatenning operator*. As can be seen in Table 4.1, Lavoisier supports all tasks in the structural category. This was expected, because Lavoisier was built on top of the *flatenning operator*, designed to address these patterns (see Section 4.4). Therefore, Lavoisier is able to deal with the different patterns that might appear in an object-oriented model.

Regarding the customisation category, three different subtasks were identified: (1) attribute filter; (2) instance filter; and (3) subclass selection. The first subtask refers to selecting which attributes of a class should be included in the flattening process. The second subtask is used to discard those instances that do not comply with a filtering criteria. Lastly, the third subtask can be employed to select and personalise a subset of the existing types in an inheritance hierarchy to intervene in the flattening process.

Partial support is currently provided for filtering instances of a class. In some cases, this filtering requires the use of arithmetical operations. We have opted for not yet including arithmetic operators, such as *additions*, *substractions*, *multiplications* or *divisions* in the language. These operators could be included with not that much effort, in the same way we added support for aggregate functions. Nevertheless, we are somehow reluctant to incorporate them because, to calculate complex cases, we might need a full-fledged programming language, which is in conflict with our initial goal of creating a language accessible to average decision makers. Despite all, we will include some basic arithmetic operators in Lavoisier as part of our future work. For the third customisation task, Lavoisier includes concrete mechanisms to select and configure the flattening of subclasses of an inheritance (see Section 4.5.2).

Concerning the calculation category, two different subtasks are considered: (1) reductions by the use of aggregation functions to summarise data, such as *average*, *min* or *count*; and (2) the creation of properties with derived values, such as a student average grades, that might not be initially included in a domain model. With respect to the first task, as it was shown in Section 4.5.4, Lavoisier provides basic support for this aggregated values. However, as it was commented, just basic spreadsheet-like functions are provided, which prevents from computing more complex aggregations using Lavoisier. In the case of derived values, Lavoisier does not provide any support for them at the current moment since, as commented, we have not included arithmetic operators in the language yet.

Summarising, Lavoisier does not seem to contain expressiveness problems, beyond those related to the computation of complex derived values to be used for instance filtering or for including new columns in an output dataset. However, this lack of expressiveness is somehow intentional in order to make the language accessible for a wider audience.

4.6.2 Conciseness and Conceptual Comparison Method

After checking that Lavoisier was free of expressiveness flaws, we analysed whether it provides advantages over representative tools of the current practice. As mentioned

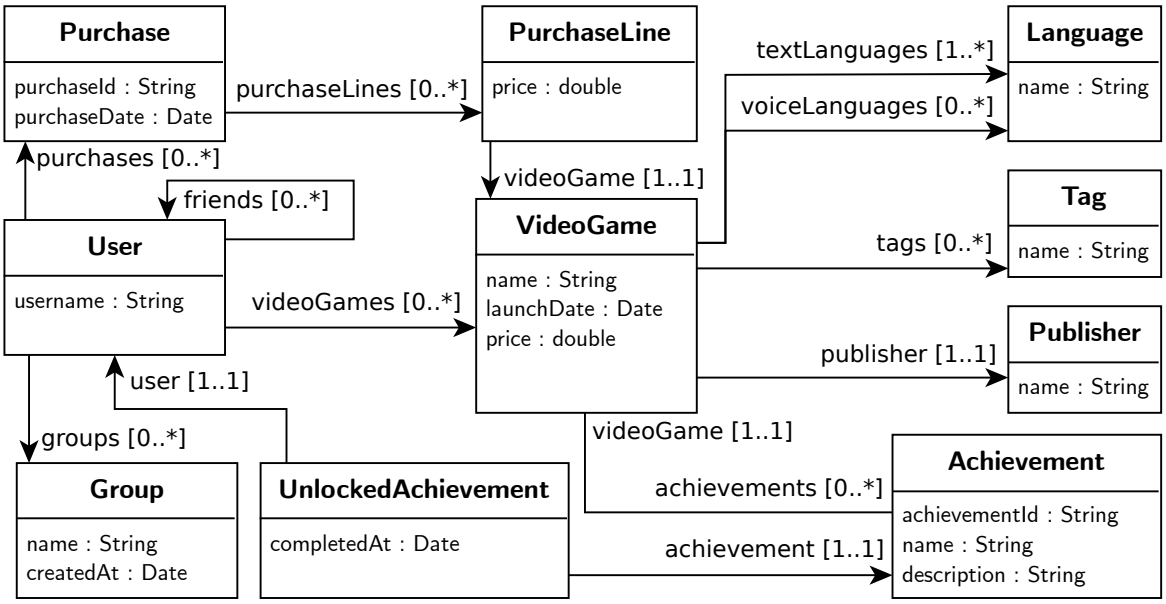


Fig. 4.17 Conceptual model of the *VideoGames* case study.

before, the selected tools for the comparison were SQL (Beighley [18]) and Pandas ([114]).

To perform the comparison, we created scripts for different dataset creation scenarios using Lavoiser, SQL and Pandas. Then, for each scenario, we measured the size of each script in characters; and we analysed what operations and parameters were required in each language. The first measurement should show how much boilerplate or low-level code is required to create a dataset by using each approach. The second comparison aims to investigate what skills are required to manage each language, as well as the conceptual complexity behind them.

The set of scenarios used for the comparison was designed to cover a wide range of cases that might appear when creating a dataset. These scenarios only address tasks related to the *structural* category. The reason for excluding the *customisation* and *calculation* categories is twofold: (1) at the time of writing this document, Lavoisier offers a limited support for the tasks of these categories; (2) Lavoisier does not claim to provide better abstractions than SQL or Pandas for these tasks.

These dataset extraction scenarios were designed over the Yelp case study. Nevertheless, this case study does not contain appropriate elements for covering some scenarios, so they were specified using a second case study, based on a online video game platform. Using a second case study also provides some evidence of Lavoisier constructs not being constrained to a concrete context, i.e., they can be applied to conceptual models from other domains.

Figure 4.17 shows a conceptual object-oriented model for a video games domain. This model represents the different elements of a video game platform, and it is inspired in existing platforms such as Steam[4]. In the video game platform, a *User* owns a collection of *VideoGames*. Users can belong to *Groups*, and maintain a list of *friends*. In addition, users have access to their video game *Purchases*. For each *VideoGame*, data are stored about its *Publisher* company, the received *Tags* (e.g. *strategy*, *multiplayer*), which in-game textual and voice *Languages* are available, and the list of *Achievements* that users can complete while playing. In the following, we refer to this example as the *VideoGames* case.

Table 4.2 shows the concrete set of scenarios that was finally used. These scenarios, as already commented, focus on discovering how the different structures that can be found in an object-oriented model are managed by each language. Each scenario was labelled with a code composed of a single character plus a number. The character indicates the kind of pattern being analysed: *a* refers to the trivial single class case; *b* are those cases where a single reference is included in the main class; *c* cases are the same as *b*'s, but for unbounded references; *d* refers to those cases involving inheritance; and *e* test the capabilities to perform multiple and multilevel reductions. Besides each scenario, the classes to be included in each dataset are provided.

Lavoisier is able to perform dataset extractions over conceptual models directly. However, SQL and Pandas need to work at the relational or table level. Therefore, we derived the relational models associated to the conceptual models of the Yelp and VideoGame case studies. This transformation process was straightforward, except for the cases where inheritance was present. Inheritances can be transformed into relational elements using typically three strategies, known as *Single Table*, *Concrete Table*, and *Class Table*, also known as *Joined Mapping* [60]. Therefore, for those scenarios that tackle inheritance, we created three different relational models, each one following a different inheritance mapping strategy; and we performed the comparison against each one of these relational models. Next subsection discusses the results of these comparisons.

Table 4.2 Dataset extraction scenarios performed in the comparison.

Code	Case	Model	Description
a	Single table/class	VideoGames	Just *VideoGame*
b1	Single-bounded reference	Yelp	*Reviews* and their *user*
b2	Single-bounded reference	VideoGames	*Achievements* and their *videoGame*
c1	Unbounded reference	Yelp	*Business* and their *categories*
c2	Unbounded reference	VideoGames	*VideoGame* and their *text languages*
d1	Inheritance	Yelp	*Features* (relational single table)
d2	Inheritance	Yelp	*Features* (relational concrete table)
d3	Inheritance	Yelp	*Features* (relational class table)
d4	Unbounded Inheritance	Yelp	*Businesses* and their *features* (relational single table)
d5	Unbounded Inheritance	Yelp	*Businesses* and their *features* (relational concrete table)
d6	Unbounded Inheritance	Yelp	*Businesses* and their *features* (relational class table)
e1	Combination (multiple)	VideoGames	*VideoGames* with *tags* and *publisher*
e2	Combination (multilevel)	Yelp	*Reviews* with their *business* and the *categories* of such business

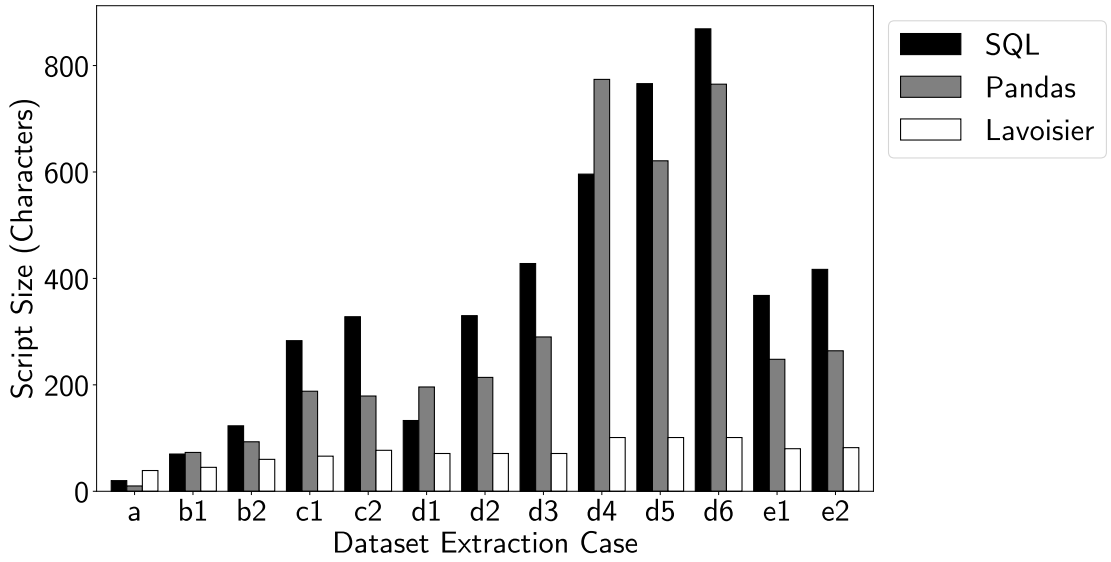


Fig. 4.18 Script size in characters of the extractions for each approach (a: single table; b: unary reference; c: unbounded reference; d: inheritance; e: combination).

4.6.3 Comparison Results

Figure 4.18 shows the size in characters of the scripts for each scenario, created using each analysed language[5]. When counting characters in each script, we ignored any kind of white space and line breaks.

On a first glimpse, it can be seen that the size of the expressions for both SQL and Pandas considerably increases as the scenario complexity grows, as it was expected. This phenomenon can be clearly appreciated in scenarios *d3-d6*, *e1* and *e2*. In contrast, Lavoisier's script size grow is steadier across all cases.

For *b*-coded cases, SQL and Pandas scripts are slightly more verbose than Lavoisier's. These cases refer to the reduction of single-value references, which is achieved by using an *include* clause in Lavoisier, a *join* operation in SQL, and a *merge* operation in Pandas. The *join* and *merge* operations are a little more complex than the *include* primitive of Lavoisier, as they require some extra parameters.

For instance, *join* and *merge* require those columns of the tables to be joined that should be used for matching rows of the two tables, whereas in the case of the *include* primitive we only need to specify the reference that should be reduced. In addition,

[4] https://store.steampowered.com/

[5] The extraction queries for SQL, Pandas and Lavoisier can be consulted in an external repository: https://github.com/alfonsodelavega/lavoisier-evaluation.

SQL and Pandas users have to manually address the following issue: column name collisions. When joining two or more tables, if these tables have the same name in some of their columns, users to manually specify an alias to differentiate them in the resulting joined table. As an example, in the *b2* extraction, the *Achievement* and *VideoGame* entities of the *VideoGames* case study are joined, but both entities have a *name* attribute. So, this pair of attributes must be aliased, either with the *as* keyword in SQL or with the *suffixes* merge parameter in Pandas. On the contrary, Lavoisier's *include* construct prevents users from having to worry about this problem by automatically adding appropriate prefixes in the reductions.

For unbounded references (*c* cases), the results for SQL and Pandas are noticeably worse. When using these two approaches, to reduce these references, we need to execute a join operation and then a pivot. So, multiple operations need to be combined along with their parameters to achieve the reduction. In the case of Lavoisier, just the name of the reference and the set of attributes to be used as pivoting attributes are required. Therefore, script size is reduced in these cases by ∼60-70%.

The performance of SQL and Pandas is also clearly worse for those cases that include inheritances (*d1*-*d6*). In general, SQL and Pandas need to perform several operations to compact the inheritance hierarchy, which adds a lot of boilerplate code in these cases. These operations are automatically carried out by Lavoisier, so the end user does not need to deal with them. The increase in script size is specially noticeable in the *d3* to *d6* scenarios. These scenarios correspond to the pattern where an unbounded reference pointing to a class included in an inheritance tree must be reduced (see Section 23). In these cases, the inheritance hierarchy needs to be compacted several times, once per leave in the inheritance hierarchy, which contributes to increase the amount of boilerplate code associated to these tasks in SQL and Pandas. In addition, it can be observed that the script size in SQL and Pandas seems to be independent of the strategy used for mapping the inheritance. Only SQL might slightly benefit from the use of the *Single-Table* mapping, as there are fewer tables to combine during the reduction.

Finally, the *e1* and *e2* scenarios are combinations of simpler cases. As it can be seen, script size increases clearly in the SQL and Pandas cases, as more intermediate operations with its corresponding boilerplate code are required; whereas script size remains stable in the Lavoisier case.

One detected benefit of Lavoisier is that reference inclusions in the dataset specifications are independent one of another. For instance, in the *e1* case, data about *VideoGames* are extracted, along with their *publisher* and *tags* references. In Lavoisier,

each reference is selected through the use of an *include* construct, and neither of these two constructs needs to be aware of the other one. On the contrary, in SQL and Pandas, one of the references would be reduced first into the video games data, and then the other one would be compacted to the result of the first reduction. Therefore, we have to pay attention to the reduction order, and to different issues such as the column name collisions problem previously commented in this section.

In summary, Lavoisier scripts' size is, generally speaking, lower than their SQL and Pandas counterparts. This size reduction can be noticeable as the number of entities to be included in a dataset grows; or when these entities are involved in inheritance hierarchies. The larger script size manifested by SQL and Pandas is caused by the need to invoke and configure lower-level data transformation operations, such as joins and pivots. These operations require extra parameters, and the combination of several operations required for some extractions results in tedious and prone-to-errors processes. Lavoisier avoids the majority of these low-level issues, mostly by the use of the defined flattening operator.

4.6.4 Threats to Validity

There are some threats to the validity of the previously commented conclusions. We discuss these threats in this section.

In the first place, it might be argued that results are due to the selection of the reduction scenarios, and that other selection might have lead to different results. These scenarios were not arbitrarily selected, but with the objective of covering all input cases that a flattening operation might face. So, we considered all possible scenarios that we might face during a flattening operation. Moreover, these scenarios were kept simple, this is, we have not created artificially complex scenarios that, according to the gathered results, would benefit Lavoisier. For instance, we have not included any scenario containing large chains of references, or very large inheritance hierarchies, which are cases where Lavoisier would have played clearly better.

Secondly, it could be considered that results are biased due to the selected case studies, and that other case studies would have returned a different outcome. The *Yelp* and *Videogame* case studies were selected just for giving some semantics to the pattern to be reduced, and for making them easier to understand. Other case studies would have lead to the same results, since the flattening operator and, consequently, Lavoisier, just processes the syntactic structure of the pattern to be flattened, so what a class or an attribute represents does not matter.

Thirdly, it can be stated that conciseness is not a good measure for easiness of use. An operator, or language, can be very concise, but really hard to use. This is absolutely true, and we have used conciseness as an indirect indicator to explore easiness of use. For each scenario, we analysed what were the reasons because Lavoisier achieved a better conciseness. The analysis revealed that conciseness was better mainly due to the use of high-level primitives that abstract complex operations and avoid large amounts of boilerplate code. Therefore, Lavoiser helps to simplify dataset creation tasks, but not because it is more concise than SQL and Pandas, but because it provides some high-level primitives specifically designed for these tasks that hide and automate different low-level operations.

Finally, it can be considered that the comparison between Lavoisier, SQL and Pandas is not fair because Lavoisier works against an object-oriented domain model, whereas SQL and Pandas do it against a relational model. Therefore, Lavoisier benefits of using a more high-level input model. This is true, and this is the reason why we decided to use an object-oriented conceptual model as input for Lavoisier.

4.7 Chapter Summary

This chapter has presented our initial contributions for allowing decision makers to participate in the data selection and data transformation steps of an analysis.

One of the identified complexities of these steps was formatting the available data into a tabular format digestible by data mining algorithms. We have formalised a flattening operator for automatically processing a selection of interconnected elements from an object-oriented domain model. This operator is able to reduce any structure pattern present in a domain model into the required tabular format.

The flattening operator has been integrated into a high-level language, called *Lavoisier*. Working with Lavoisier, users just specify, through a set of high-level primitives, which part of a domain model should be considered for a data mining task, and the language automatically rearranges the selected data into an appropriate tabular format. This avoids that large and complex scripts to accomplish this task have to be created by hand, saving time and reducing errors.

We compared Lavoisier against state-of-the-art solutions for the preparation and arrangement of data. This comparison involved the specification of dataset extractions with Lavoisier and two representatives of the state of the art in data transformation and manipulation, i.e., SQL (Beighley [18]) and Pandas (McKinney [114]). From this comparison, it can be stated that Lavoisier specifications are more compact than the

others, and the constructs offered by this language are more abstracted from low-level details, thanks to the usage of the flattening operator.

As another result of this comparison, we detected that Lavoisier's syntax was not adequate for the definition of complex derived values, as this syntax did not include low-level features such as arithmetical operations. We will include support for these operations in the future, while trying to avoid increasing the entry-level complexity of the language syntax. At the same time, we will perform empirical experiments, to see how comfortable decision makers are when using Lavoisier for data preparation tasks.

Chapter 5

Pinset: Advanced Extraction of Datasets from Models

5.1 Introduction

To evaluate Lavoisier, we tested it in the educational and software engineering domains. We selected these domains because:

1. They are domains in which we have expertise.
2. Data mining has demonstrated to be helpful in these domains (D'Ambros et al. [40], Jugo et al. [84], Omta et al. [127], Zorrilla and García-Saiz [176]).
3. In the particular case of software engineering, there is a growing interest for using data analysis techniques in the model-driven engineering community (Babur et al. [9], Di Rocco et al. [48]), which is very familiar to us.

During this evaluation, we identified a limitation of Lavoisier: it was complex, or directly unfeasible, to create datasets containing columns that hold complex derived values. For instance, in the educational domain, complex values such as the *"percentage of completed assignments for those subjects that were failed"* can be calculated for each student. This kind of values are useful to domain experts as indicators of certain issues or behaviours. As an example, the previous indicator might be used to know whether students made an effort to pass the subjects but failed or, instead, they dropped early.

Calculating these complex values could be supported in Lavoisier by adding some constructs to specify basic computations, such as conditionals or loops. Neverhtehess, this idea is in conflict with one of the main objectives of this language, i.e., that

Lavoisier primitives are accessible to any potential user. Some of these users, like general teachers, are very likely to not have programming skills. So, even if we include these constructs in Lavoisier, they would not be able to take advantage of them. Moreover, adding these constructs to Lavoisier's syntax might pollute it, and increase its learning curve. Therefore, we decided to not include them into Lavoisier and leaving it as it is.

On the other hand, domain experts from other domains, such as software engineers or physicists, who are expected to have some programming skills, could benefit from the availability of these constructs. Thus, we decided to create an alternative language to Lavoisier, called *Pinset*, oriented to these power users.

Pinset is implemented as an extension of the Epsilon (Paige et al. [128]) model-driven development suite. This allows making use of the facilities this platform provides, such as the OCL-like expressions provided by *EOL (Epsilon Object Language)* (Kolovos et al. [98]) or support for many model types, increasing the ability to query models from the object-oriented domain models that are supported by Lavoisier.

We evaluated Pinset to confirm that this language is able to overcome the Lavoisier limitations that originated its development. In addition, we compared Pinset with a general purpose transformation language for the objective of extracting datasets from models. As a result of this comparison, Pinset scripts were ∼70% more compact, more readable, and easier to maintain.

The rest of this chapter is organised as follows. Section 5.2 details the motivation behind this work. Pinset's syntax constructs are described in Section 5.3, while Section 5.4 presents the structure and internals of this language. Then, in Section 5.6 we describe the evaluation activities we performed to assess pinset capabilities. Lastly, Section 5.7 summarises and concludes this chapter.

5.2 Motivation: Support for Advanced Calculations

This section details the motivation behind the creation of Pinset, based mostly in some shortcomings we found in Lavoisier when performing calculations of derived data. First, we describe the running example that is used throughout the chapter, and then we introduce the shortcomings we found in Lavoisier.

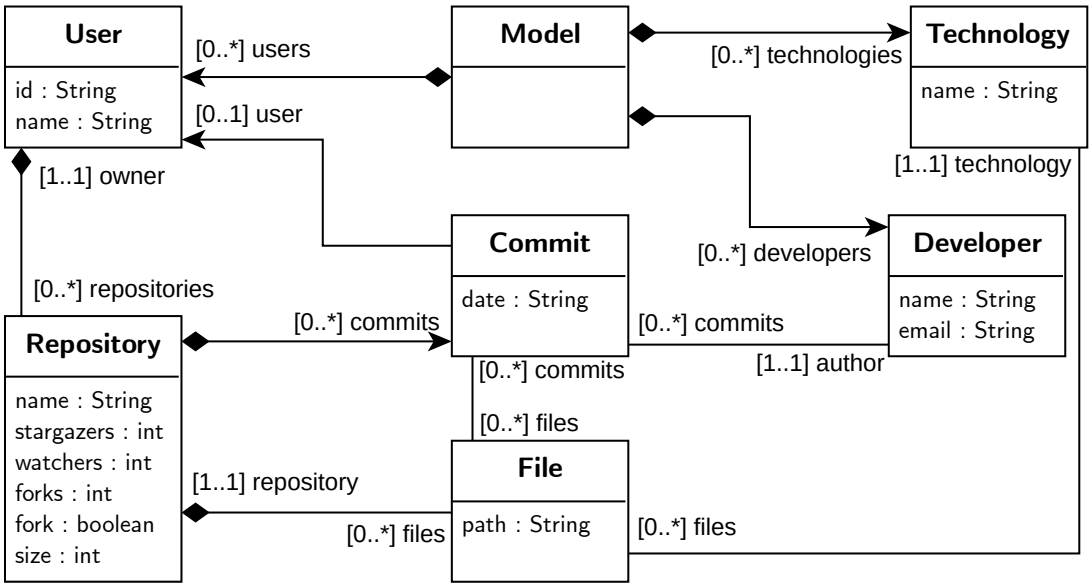


Fig. 5.1 The Github-MDE (Ghmde) model.

5.2.1 Running Example: Github-MDE (Ghmde)

Kolovos et al. [101] carried out an assessment study on the usage of model-driven technologies in open-source software projects. For this purpose, they performed an extensive search over versioned repositories hosted on GitHub[1], where they looked for files of several model-based technologies (e.g. EMF, ATL, Xtext, Epsilon). The collected files information was later enriched with more data about the repositories, commits and developers who performed those commits. In order to make this information available to other researchers, they generated and published the *Github-MDE (Ghmde)* model, which contains data about 1928 repositories, 34.442 files and 74.403 commits.

The domain model to which the Ghmde data conforms can be found in Figure 5.1. A *Repository* is composed of *Files* and *Commits*, which are used for registering file inclusions and/or modifications. Each file belongs to a concrete *Technology*, such as *Ecore*, *ETL*, or *Xtext*, among others. As stated by Kolovos et al. [101], Github differentiates between the *User* that uploaded a commit to the repository and the *Developer* that actually authored this commit. So, these two entities exist separately in the domain model.

[1] https://www.github.com/

5.2.2 Limitations of Lavoisier

In our efforts of making the Lavoisier language easy to use by people without experience in data mining techniques or even in programming languages, we did not include in its syntax some advanced constructs that may be required to perform calculations of derived data. Some examples of these constructs are the following:

- ***L1*: Not instance-based datasets**. Lavoisier's syntax is based on selecting a type, i.e., the *main class* of a domain model, and then gather information around this type. Instances of this type become the rows of the resulting dataset. In some cases, we might want to aggregate data from a model instead of focusing in a concrete type. For instance, in the Ghmde example, we could want to extract a dataset that aggregates the information of commits and developers pear year. Examples of other aggregates might be the number of commits by Technology, the most popular repositories, or the developers that contributed most. Each row of this dataset would show the mentioned data for a concrete year, e.g., 2018 or 2019. Therefore, rows of this dataset do not represent concrete instances of any type from the input model.

- ***L2*: Arithmetic operations**. Lavoisier does not allow performing arithmetic operations such as additions or divisions, which might be useful to calculate derived values. As an example, we could be interested in extracting a dataset of *Repositories* data including a column that represents the ratio of *stargazers*[2] over *watchers*[2] for each repository.

- ***L3*: Support of advanced operators**. In addition to the previous example, typical first-order logic operators (e.g. select, collect, exists) that are available in other model-oriented languages such as *OCL* (Object Management Group [124]) or EOL (Kolovos et al. [98]) are not supported by Lavoisier. Including these operators could be interesting, more so if end users are familiar with them. For instance, if we had a collection of *Commits*, and we were interested in the set of *Technologies* of the *Files* affected in those commits, we could concatenate some of these operators to obtain this information. In EOL, such calculation could be performed with the following operation: *commits.collect(c | c.files).flatten().collect(f | f.technology).asSet()*. This operation might be perceived as natural by experienced programmers, but it can be scary for people without these skills. Therefore, this kind of expression is not supported in Lavoisier.

[2]These terms are specific to the Github platform. *Stargazers* are users who have starred a repository, while *watchers* are the ones who follow the evolution of such repository.

- ***L4*: Availability of imperative code structures**. The use of a declarative syntax makes Lavoisier's constructs natural and easy to use. However, some advanced calculations might be better performed through the use of imperative code structures, i.e., by employing code blocks based on conditions, loops and variables. For example, we could be interested in analysing the commits of each *Developer* during the last year, for which we would first extract a dataset with this information. In this dataset, we could want to add a column including the most contributed repository for each developer. This column would be calculated by gathering the commits from the last year, grouping these commits by the repository to which they belong, and then selecting the repository with the greatest number of commits. This operation, although not impossible to achieve with a declarative syntax, might be easier to achieve if split into different subtasks Therefore, we would be confronting an easier problem in each of these subtasks, and we could test these tasks separately.

- ***L5*: Reuse of intermediate calculations**. The existence of variables could also benefit performance. For instance, if we wanted to generate several datasets containing data from a subset of repositories obtained by using a filtering operation, having access to this subset through a variable would make unnecessary to recalculate it each time, which would translate in a performance improvement.

In summary, Lavoisier's declarative syntax does not support certain features that may be required to perform some advanced calculations during dataset creation, such as complex aggregations. We wanted to offer support for these calculations but, at the same time, we did not want to transform Lavoisier into a language only accessible for programmers. Therefore, we decided to build a new language for these specific power users. This new language was denoted as *Pinset (PINcer of dataSETs)*, and it is presented in this chapter. Next section describes this language through dataset extraction examples performed over the Ghmde model presented in Section 5.2.1.

5.3 Solution Description

We start by presenting the syntax of Pinset with a basic example, followed by more in-depth descriptions of some advanced features.

5.3.1 Syntax Overview

Listing 5.1 shows an example of Pinset script. This script extracts some basic information of *Repositories* from data conforming to the Ghmde model of Figure 5.1.

Listing 5.1 A Pinset dataset rule that extracts some basic information of a Repository.

```
1  dataset repositoriesInfo over r : Repository {
2    column name : r.name
3    column owner_name : r.owner.name
4    column owner_id : r.owner.id
5    column number_of_commits : r.commits.size()
6    column owner_commits_ratio {
7      if (number_of_commits == 0) {
8        return 0;
9      }
10     return r.commits.select(c | c.user = r.owner)
11                     .size()
12                     / number_of_commits;
13   }
14   column files_with_several_commits : r.filesWithMoreCommitsThan(1)
15                                         .size()
16 }
17
18 operation Repository filesWithMoreCommitsThan(numCommits: Integer) {
19   return self.files.select(f | f.commits.size() > numCommits);
20 }
```

A Pinset script defines one or more *dataset rules*. Each one of these rules specifies the structure and contents of a different dataset. In its simplest form, a dataset rule consists of (i) a name, (ii) a typed parameter, and (iii) a set of column generators. Listing 5.1 provides a dataset rule example, denoted as *repositoriesInfo*. The typed parameter specifies which entity of the input model is going to be processed when populating the rows of the output dataset. In the following, we will refer to the entity selected by this parameter as the *main entity* of the dataset rule. For instance, in Listing 5.1, this main entity is *Repository*. Lastly, column generators are used to define the columns that the final dataset will have, and how the values of these columns are calculated.

As output, the execution of a Pinset script generates a CSV file for each specified dataset rule. Each generated file has the same name as its corresponding rule. When executing the example of Listing 5.1, a *repositoriesInfo.csv* file is created. The first row of this file contains the name of the defined columns separated by commas. Then, a row

is included for each element of type *Repository* in the input model. Each row contains the column values calculated by executing the column generators of the dataset rule, being these values separated with commas.

Column generators can have different flavours. In this first example, the *Column* generator is employed. This generator requires a *name*, which defines the header of the column to be generated; plus a piece of code that specifies how this column must be calculated for each instance of the main entity. This piece of code can be defined using different styles. In the simplest version, column values are obtained through an *EOL* expression. *EOL (Epsilon Object Language)* (Kolovos et al. [98]) is an OCL-like language from the Epsilon (Paige et al. [128]) model-driven development suite. This language has capabilities for manipulating models conforming to a metamodel structure. These EOL expressions are invoked over each instance of the main entity.

Listing 5.1, line 2 shows a *Column* generator example. This example specifies that the output dataset has a column called *name*, which contains the name of each repository, retrieved through the expression *r.name*, where *r* represents each concrete instance being processed. This variable was specified in the header of the dataset rule, as the parameter name (line 1).

When defining a column, it is possible to traverse the references of the main entity. For instance, the second and third column definitions (lines 3 and 4 respectively) define the columns *owner_name* and *owner_id* by navigating the *owner* reference of a repository (i.e. *r.owner.name* and *r.owner.id*).

Additionally, we can invoke functions from the processed objects, such as the operations offered by EOL to manipulate collections. In line 5, the column *number_of_commits* is calculated by invoking the *size* function of the *commits* collection (*r.commits.size()*).

As commented, more complex expressions can be employed if needed. The column *owner_commits_ratio* stores the ratio of commits that the owner of a repository has uploaded over the total number of commits in the repository. Such a ratio can be used to estimate the amount of external collaboration that a project's repository has attracted. This column is calculated with an EOL block (lines 6-13), which is composed of a set of instructions that return the value that will be used to populate the column. If the repository has no commits, we return zero without performing any operation (lines 7-9). When the repository has commits, we calculate the owner ratio by dividing the number of owner commits by the total number of commits (lines 10-12).

Column values are calculated in the same order they appear in the dataset rule. This way, already calculated columns can be used in posterior column definitions. For

instance, when defining *owner_commits_ratio*, we referred to the *number_of_commits* column previously defined (lines 5-13).

Lastly, the column definition of lines 14-15 show that it is possible to call external functions from column expressions. The column *files_with_several_commits* stores the number of files from the repository that have more than one commit. To calculate this number, the *filesWithMoreCommitsThan* function is called (lines 18-20) with 1 as a parameter. This function returns those files of a repository whose number of commits is greater than the one provided as a parameter.

After this initial overview, the following sections describe more advanced data-extraction mechanisms provided by the language.

5.3.2 Properties Accessors

When we want to define columns that only hold values from properties of the main entity, the *Column* generator syntax can become too verbose and redundant. For example, in the *name* column definition of Listing 5.1 (line 2), the name of the column matches the name of the retrieved property. Therefore, this name could be easily deducted from that property. For these cases, Pinset provides shorthand constructs that allow defining columns for simple properties in a more concise way.

Listing 5.2 Dataset extraction that employs *properties* and *reference* helpers.

```
1 dataset repositoriesInfo over r : Repository {
2   properties[name, watchers, stargazers as stars]
3   reference owner[name, id]
4 }
```

Listing 5.2 shows how this syntactic sugar can be used. In line 2, the *properties* generator selects some properties from the processed type to be included as columns. Property names must be indicated between square brackets, separated by commas. For each property, a new column with the same name is created, which holds the value of that property for each processed element. These properties must hold values of a primitive type, not being possible to apply this generator over references to other entities. In our example, the *name*, *watchers*, and *stargazers* properties of a Repository are included in the target dataset. Properties can be renamed if desired, by using the *as* keyword. In the example, the *stargazers* property, which comes from the Github terminology, is renamed to *stars*, which is more understandable for the general public.

To include some information about types related to the processed one, we can use the *Reference* generator. This construct receives the name of a reference of the

processed type, and a set of properties from that reference. The generator creates a new column for each specified property of the reference, and the values of these columns will be simply obtained from the corresponding properties, as before. If no alias is provided, the name of the columns is obtained by combining the reference and property names with an underscore. In our example, this construct is used to include the *name* and *id* of the owner of each repository (Listing 5.2, line 3). As a result, new columns denoted *owner_name* and *owner_id* are created. This generator can only be used over references with an upper bound of 1. For processing those references with a greater upper bound, we must use a different generator.

The presented generators automatically manage any presence of *null* values in the model. If a property is not present, a blank value is inserted instead. In the same way, if the reference of an element points to null, blanks are inserted for all included properties of that reference.

5.3.3 Row Filtering Options

In the previous examples, datasets contain one row for each instance from the main entity. However, it could be the case that we are not interested in processing all these instances, but a subset of them. Pinset offers two alternatives to perform instances filtering: (1) specifying a *guard* condition; or (2) declaring a *from* expression. One alternative might be more suitable than the other, depending on the characteristics of the filtering process. Listings 5.3 and 5.4 illustrate both alternatives, respectively. For the sake of simplicity, column definitions have been omitted from these listings.

Listing 5.3 Selection over the type elements of a dataset with a *guard*.

```
1 dataset developerSelectionGuard over d : Developer {
2   guard : d.commits.size() > 25
3   ...
4 }
```

A *guard* is a condition declared with the *guard* keyword followed by a boolean expression (Listing 5.3, line 2). This expression is evaluated over each instance of the corresponding type. Those instances that do not match the condition are discarded. Therefore, no row is generated for them in the dataset. As an example, in Listing 5.3, we are generating a dataset with *Developers* data. However, as specified by the guard (line 2), only those developers who have authored more than 25 commits are included in the dataset.

Listing 5.4 Selection over the type elements of a dataset with *from*.

```
1 dataset developersSelectionFrom over d : Developer
2 from : Developer.all.select(dev | dev.commits.size() > 25); { ... }
```

Another way of performing the same selection is shown in Listing 5, which employs a *from* clause. This clause explicitly indicates the collection of elements to be used in the creation of the dataset. The clause is declared with the *from* keyword, followed by an expression that returns the mentioned collection of instances. Listing 5.4 provides an example of this use case. The from expression includes those developers with more than 25 commits (line 2).

It should be noted that, when the from clause is used, the dataset rule may not need to access the input model to search for instances of the main entity if, for instance, a collection of instances has been previously calculated. In Pinset, it is possible to perform these prior calculations in *pre* blocks, which are sets of EOL statements that are executed before any of the dataset rules. Any variable defined by these blocks is visible and ready to be used in the these rules. To illustrate these blocks, Listing 5.5 repeats the example of Listing 5.4, but this time the calculation of the developers subset takes place in a *pre* block.

Listing 5.5 Selection over the type elements of a dataset with *from*.

```
1 pre {
2   var significantDevs =
3     Developer.all.select(d | d.commits.size() > 25);
4 }
5
6 dataset developersSelectionFrom over d : Developer
7   from : significantDevs { ... }
```

In the pre block of the example, the *significantDevs* variable includes those developers that have authored more than 25 commits (lines 2-3). Then, in the *from* expression of the dataset rule (line 6), just a reference to this *significantDevs* variable is necessary.

Therefore, in those cases where the same subset of instances is used as input for several dataset rules, the *from* option might be preferred over a *guard* for performance reasons, since this subset would be calculated just once for all rules.

Finally, it is worth pointing out that both mechanisms are not mutually exclusive, and can be applied in combination. When combined, the guard condition is evaluated over the collection of elements provided by the from clause, so the guard is used in this case to filter that collection. This feature might be used to obtain refined datasets after performing some preliminary analysis over a broader dataset. For instance, a first analysis over all developers with more than 25 commits might indicate that those

Table 5.1 Dataset storing usage of technologies in a repository.

name	uses_Ecore	uses_GMF	uses_Xtext	...
R_1	true	false	true	...
R_2	true	true	false	...
...	...	...	...	...

using certain MDE technology, e.g., *Ecore*, participate in more projects than others. A second analysis could focus just in this more concrete subset of developers, in order to investigate this phenomenon more accurately. Such an example is shown in Listing 5.6.

Listing 5.6 Selection by combination of *guard* and *from* expressions.

```
1  pre {
2    var significantDevs =
3      Developer.all.select(d | d.commits.size() > 25);
4  }
5
6  dataset developersSelectionFromAndGuard over d : Developer
7  from : significantDevs {
8    guard : d.commits.collect(c | c.files)
9                     .flatten()
10                    .collect(c | c.technology)
11                    .exists(t | t.technology.name = "Ecore")
12   ...
13 }
```

5.3.4 Multiple Columns Definition: Grid

In some cases, we detected that some columns were defined with an almost identical expression. This happens, for instance, when we want to determine which technologies are used in each repository. This dataset would look like shown in Table 5.1.

The columns of this dataset represent the different MDE technologies that might appear in each repository. Each row of this table represents a repository, and the cells of these rows determine the usage of technologies, i.e., if certain technology is used in a repository, then its corresponding cell would be *true*, and *false* otherwise. For instance, the repository *R_1* of Table 5.1 uses the *Ecore* technology, but it does not use *GMF*.

As it can be expected, the EOL expression for computing the *uses_Ecore*, *uses_GMF* or *uses_Xtext* columns is pretty similar. So, the one-by-one definition of these columns by using the *Column* generator becomes redundant, as the expression that checks

the usage of a technology is the same except for the concrete technology that is searched each time. Listing 5.7 shows a dataset using this strategy, only including the 3 mentioned technologies (there are 11 in the GhMDE model). In the example, we look for the existence of technologies in the repository with the same name as the one for each column. We obtain the technologies used in a repository through the *getUsedTechnologies* function (lines 1-4). The *@cached* annotation of this function improves the performance of this operation by storing the calculated values to avoid future calculations.

Listing 5.7 Technologies usage by each repository using the *column* generator.

```
1  @cached
2  operation Repository getUsedTechnologies() {
3    return self.files.collect(f | f.technology).asSet();
4  }
5
6  dataset technologiesInRepository over r : Repository {
7    properties [name]
8    column uses_Ecore : r.getUsedTechnologies()
9                          .exists(t | t.name = "Ecore")
10   column uses_GMF   : r.getUsedTechnologies()
11                         .exists(t | t.name = "GMF")
12   column uses_Xtext : r.getUsedTechnologies()
13                         .exists(t | t.name = "Xtext")
14   ...
15 }
```

In addition to the redundancy problem in the definition of columns, the dataset rule of Listing 5.7 would require updates if, for instance, new technologies appear in subsequent data bundles. If that is the case, we would need to add extra column generators to process these new technologies adequately.

The column expressions of Listing 5.7 can be abstracted by converting their variable element, i.e., the concrete technology to be searched, into a parameter of certain construct. With this purpose, we developed the *Grid* generator, which allows defining multiple columns over the same expression. This generator creates a set of columns based on a collection of elements, denoted as *keys*, which is specified by means of an EOL expression. Each key is then processed to generate a column, based on two extra components: (1) a *header*, which determines the name of each column being generated: and, (2) a *body*, which contains the piece of code that calculates the value for each generated column. Both header and body expressions can access to the key being processed through the *key* reserved word.

Listing 5.8 Grid example that generates the dataset of Table 5.1.

```
1  @cached
2  operation Repository getUsedTechnologies() {
3    return self.files.collect(f | f.technology).asSet();
4  }
5
6  dataset technologiesInRepository over r : Repository {
7    properties [name]
8    grid {
9      keys : Technology.all
10     header : "uses_" + key.name
11     body : r.getUsedTechnologies().includes(key)
12   }
13 }
```

Listing 5.8 shows a dataset rule that uses a grid generator (lines 8-12) to create the dataset of Table 5.1. In this case, the keys collection is specified by obtaining all the diferent elements of type *Technology* that exist in the input model (line 9). These elements represent all existing technologies from the input model. Next, the *header* specifies the name of each column by concatenating the *"uses_"* string with the *name* property of each key (line 10). Lastly, for each instance of the main entity being processed, the *body* is evaluated for each *key*. In our example, this body is used to determine whether a technology is used in a repository (line 11). This body uses again the *getUsedTechnologies* function to retrieve the technologies used in a repository (lines 1-4).

It should be noticed that a grid can be used to generate datasets with a variable number of columns, i.e., dependent on the contents of the input model. For instance, for a concrete model, maybe only a reduced subset of technologies is used, e.g., four or five technologies. On the other hand, we could use another model containing a very heterogeneous set of repositories, including more than 20 technologies. The grid generator would work in the same way in both cases, generating on each case a different number of columns. So, those dataset rules that employ grids avoid needing updates in their column definitions depending on the input data, as opposed to the example of Listing 5.7. To this respect, the grid generator is somehow similar to the pivot operation described in Chapter 4.4.2.

5.3.5 Nested Column Definitions

Sometimes, a subset of generators of a dataset rule use the same value to calculate their columns. This value might be derived, i.e., obtained by executing a set of expressions over the processed data. In this context, to prevent code duplication, we might encapsulate these expressions into an external function. Moreover, if we wanted to avoid that this value is calculated repeatedly, we can make use of the *cached* function annotation provided by EOL.

However, the excessive use of external functions might pollute the Pinset script with a lot of dataset-specific operations. To alleviate this, we offer the *NestedFrom* composite generator. This generator can be understood as a way to define on-the-fly anonymous functions inside dataset rules. The result of these functions is stored in a variable, which can be used by several column generators.

Listing 5.9 Example of a *NestedFrom* definition including internal generators.

```
1  @cached
2  operation Collection getUsedTechnologies() {
3    return self.collect(commit | commit.files)
4               .flatten()
5               .collect(file | file.technology)
6               .asSet();
7  }
8
9  dataset nestedFrom over d : Developer {
10   properties [name, email]
11   from recent_commits : d.commits.select(c |
12                                          c.date.getYear() == 2014) {
13     column number : recent_commits.size()
14     grid {
15       keys : Technology.all
16       header : "use_" + key.name
17       body : recent_commits.getUsedTechnologies().includes(key)
18     }
19   }
20 }
```

Listing 5.9 shows an example of this generator. In the example, a dataset with data from developers is created. This dataset gathers information of the latest commits available, which in the case of the Ghmde model corresponds to the year 2014. Therefore, the collection of these latest commits is used in the generation of different columns.

To avoid code duplication and having to define external functions, we made use of a *nestedFrom* generator (lines 11-19).

A *nestedFrom* is composed of a *variable* declaration, an *expression* and a *block*. The block contains a set of column generators. The expression computes a value that is stored in the variable. This variable can then be used by any column generators defined inside the block.

In the example, the *recent_commits* variable stores the result of the expression that follows (lines 11-12), which gathers all commits of the developer in the year 2014. Two column generators are defined inside the *NestedFrom* block (lines 13-18). First, a *Column* generator gathers the total *number* of recent commits of the processed developer by calling the *size* function over the *recent_commits* variable. Then, the MDE technologies used in these recent commits are obtained through a *Grid* generator, in a similar way as to the example shown in Listing 5.8. To improve the clarity of the code snippet, some functionality has been again extracted into the external function *getUsedTechnologies* (lines 1-7).

The columns inside a *NestedFrom* block are calculated as usual, but taking into account that the defined variable is available for the nested column generators of the block. Column names of these nested generators are prefixed with the *NestedFrom* variable name and an underscore. Therefore, the two generators inside the *NestedFrom* of Listing 5.9 define the following columns: *recent_commits_number*, and one column for each technology indicating its usage by the developer, e.g., for the Ecore technology, a column *recent_commits_use_Ecore* would be generated.

5.3.6 Typeless Dataset Rules

In the previous examples, datasets were created over an entity from the model, e.g., *Repositories* or *Developers*. However, it is possible that, instead of placing the data from each instance of a type in a row of the final dataset, we may want to aggregate instances of certain types according to some grouping criteria. For instance, we might be interested in knowing the number of commits by year across all technologies and repositories, so that we can see the tendency on the use of MDE technologies in open-source software throughout the time. Similarly, we could also obtain the number of different developers in each year. So, to obtain this dataset, we would calculate aggregations for each existing year in the data. In the case of the Ghmde model, this range goes from 2003 to 2014.

To perform this kind of aggregations, Pinset provides *typeless dataset rules*. These rules do not have a main entity. Instead, they iterate over a set of elements gathered

by a *from* clause. In section 5.3.3, the *from* construct was presented as a row filtering mechanism, where it provided the list of instances of the main entity that were to be transformed into rows. The process now is the same: the *from* expression provides the list of elements used to generate rows, but these elements are not restricted to an entity from the model.

Listing 5.10 Typeless rule that counts the number of commits and developers per year.

```
1  operation String getYear() : Integer{
2    return self.substring(0,4).asInteger();
3  }
4
5  @cached
6  operation commitsByYear(year : Integer) {
7    return Commit.all.select(c | c.date.getYear() == year);
8  }
9
10 dataset infoByYear over year from : 2003.to(2014) {
11   column year : year
12   column num_commits : commitsByYear(year).size()
13   column num_devs : commitsByYear(year).collect(c | c.author.email)
14                                        .asSet()
15                                        .size()
16 }
```

Listing 5.10 shows the rules that generates the dataset of the previously described example (lines 10-16). This rule uses a collection of years, i.e., integers, as elements to generate the rows of the dataset. The parameter of the rule, which does not have a type in this case, holds the *year* value (line 10). This parameter will iterate over the values provided by the expression *2003.to(2014)*, which returns a sequence with all the integer values between 2003 and 2014, including the limits. We used this expression for the sake of simplicity of the example, as it would have been more appropriate to extract this collection of years from the commits of the model.

The result of this rule is a dataset containing twelve rows (one for each year in the interval) and three columns: the year being considered, the number of commits, and the number of different developers in that year. The number of commits is calculated with the help of the function *commitsByYear* (lines 5-8), which returns the collection of commits of the year received as a parameter. The date of a commit is stored in the Ghmde model as a string. So, to extract the year, we need to parse a fragment of this string, which we do with the function *getYear* (lines 1-3). The number of different developers is obtained also by calling to *commitsByYear*, and then obtaining the set of

unique developers from the commits of each year based on the developer emails, in the same way that Kolovos et al. [101] did in their work.

5.3.7 Column Post-Processing

After a dataset is generated, we might need to perform some extra computations to adopt the dataset to the particularities of a specific data mining algorithm. For instance, some data mining algorithms do not allow null/missing values. Therefore, before using these algorithms, we must process the dataset and replace nulls with a proper default value, such as the column mean or mode.

To support this kind of transformations, Pinset provides with a set of constructs denoted as *post-processors*, which are specified through column annotations. Listing 5.11 shows two of these post-processors.

Listing 5.11 Post-processing nulls filling and normalization examples.

```
1 dataset postProcessing over r : Repository {
2   properties [name]
3   @fillNulls unknownId
4   column owner_id : r.owner.id
5   @normalize
6   column stars : r.stargazers
7   @normalize
8   column number_of_commits : r.commits.size()
9 }
```

The first post-processor, *fillNulls* (lines 3-4), replaces null values with a specified one. In the Ghmde model, the Github Id of the repository owners was not present for about 30% of the *User* instances. Using the *fillNulls* post-processor, these null values are replaced with an "unknownId" generic value.

Other supported ways of filling nulls can be applied: using *mode* or *mean* as value would fill null cells with the mode or the mean of the column, respectively. Obviously, the *mean* option can only be used in those columns that are numerical.

The second post-processor helps with another constraint typically imposed by data mining algorithms: normalising numerical vales of a column into the [0, 1] interval. This normalisation is important when comparing numerical values of different scale, such as number of commits and number of stars of a repository extracted in the example (lines 5-8). The *normalize* post-processor is used to carry out this transformation for the the *stars* and *number_of_commits* columns.

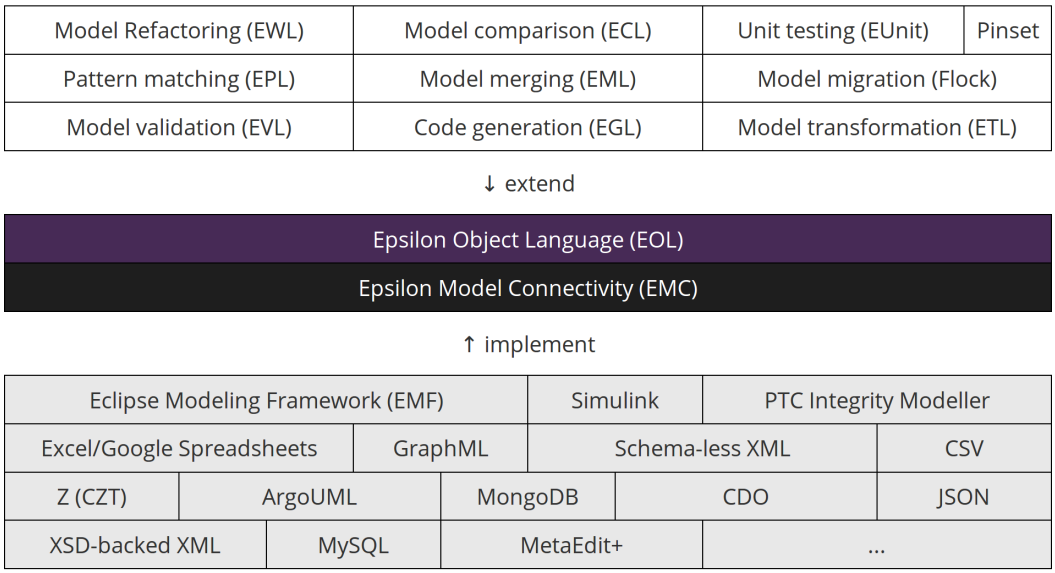


Fig. 5.2 Epsilon architecture: languages (top) and technologies (bottom).

This last feature finishes our description of the Pinset language. The next section gives details about how we implemented Pinset.

5.4 Implementation

Pinset is freely available as open-source software[3]. The development is active and new features might be added in a near future. Pinset is built in top of the Epsilon model-driven development suite (Paige et al. [128]). Right now, the implementation consists of two Eclipse plugins. The first one contains Pinset's parser and execution engine, while the second one offers an Eclipse editor with support for Pinset's syntax and configurable execution wizards.

The following sections describe the internal components of Pinset, and the steps that take place in the execution of a Pinset file.

5.4.1 Epsilon Platform Usage

As commented, Pinset is build atop Epsilon. Epsilon (Paige et al. [128]) is a software suite composed of interoperable languages, each supporting a different model management task. Some examples of this languages are: (1) EVL () for model validation; (2) Flock () for model migration; (3) ETL (Kolovos et al. [99]) for model-to-model

[3] https://github.com/alfonsodelavega/pinset

transformations; or (4) EGL (Rose et al. [141]) for model-to-text transformations. All these languages share a common *core*: the *Epsilon Object Language (EOL)* (Kolovos et al. [98]). This language provides OCL-like expressions for model management, and supports imperative language structures such as conditional and loop statements, user-defined operations and import declarations. Figure 5.2, top shows some of the languages that have been defined based on EOL. All languages by the Epsilon suite are developed atop EOL's syntax and execution engine. Following the same approach, we implemented Pinset using EOL as base.

The syntax of EOL is defined in an ANTLR grammar, and any program written in EOL is executed through a Java interpreter. We defined our own grammar that made use of those EOL's elements that were useful for the objectives of Pinset, and extended the interpreter to be able to execute Pinset scripts.

Other benefit provided by Epsilon is that all its languages can be used to process information stored in multiple formats, such as EMF, UML, XML, or spreadsheets, among many others. A subset of these formats are shown in the bottom of Figure 5.2. This access is provided by the *Epsilon Model Connectivity (EMC)* layer, which allows supporting new model types through the implementation of a driver.

5.4.2 Structure of Pinset

Figure 5.3 shows the abstract syntax of Pinset. As the language is defined over EOL, some elements are inherited from it, such as *Expressions* or *Operations*.

Pinset scripts are organized in modules. A module (*PinsetModule*) can import external modules from the Epsilon platform, such as an EOL library file with operation definitions. Each module also contains optional *Pre* and *Post* EOL statement blocks, which are executed before and after the datasets are generated, respectively. As we have seen, it is also possible to declare *Operations* for the encapsulation and reuse of common functionality during the dataset creation process.

Additionally, the module contains information about where and how to store the generated datasets. It requires an *outputFolder*, an *extension* for the dataset files, and the *separator* to be placed between the columns. By default, CSV files are generated, but these output settings can be modified.

The main component of a module are its *DatasetRule* definitions. These rules, as previously defined, have a *name*, a *parameter*, and a set of *ColumnGenerators*.

The *ColumnGenerator* interface defines two methods: *getNames*, which returns the names of the columns it defines; and *getValues(Object)*, which calculates the column values for the object that is passed as parameter. Depending on the column generator,

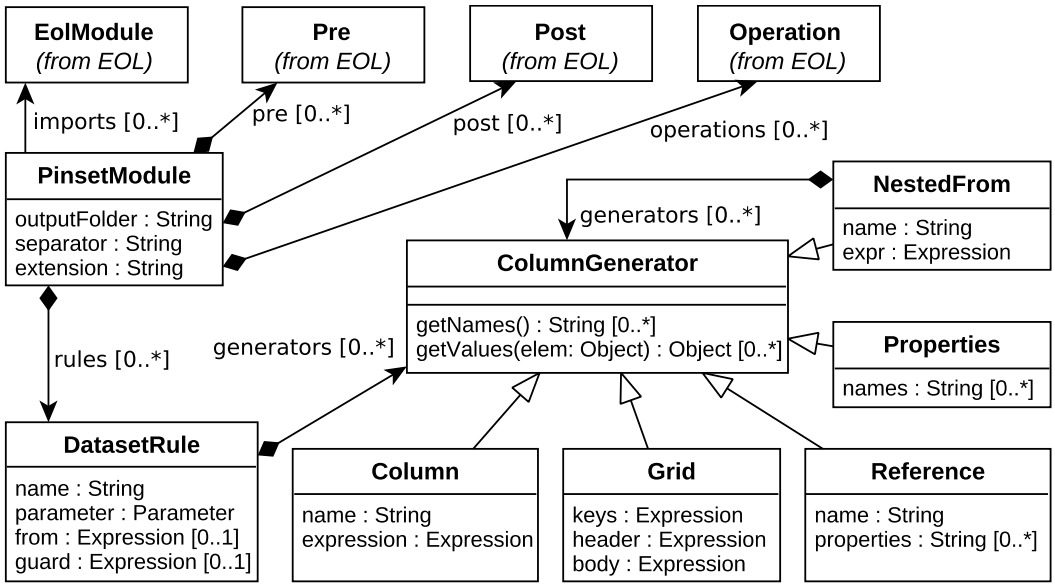


Fig. 5.3 Abstract syntax of Pinset.

one or more columns will be generated. For instance, a *Column* construct always returns one column, while in the case of the other generators this number is variable.

5.4.3 Execution Process of a Pinset Script

To process a Pinset script, first, *pre* blocks are executed, in the same order they were declared. Secondly, each defined *DatasetRule* is processed individually. To do it, the elements that will be iterated to generate rows are gathered as a first step (see section 5.3.3). Then, column names are obtained once from the *getNames* method of the generators. These names define the header of the resulting dataset. Then, the selected elements are processed one by one. Cell values of a row are calculated by feeding the *getValues* method of the declared generators with the respective element of that row. For those generators that employ expressions, the element is made accessible through the name of the rule's *parameter*. Post-processing operations are performed as the last calculation step. The obtained datasets are stored following the output details of the *PinsetModule*, regarding destination folder, column separator and file extension. Finally, *post* blocks, are executed, in the order that they were declared. These blocks share the same structure as the *pre* blocks we introduced in Section 5.3.3 but, as their name indicates, their code is run at the end of a Pinset script execution instead of at the beginning.

5.5 MDE that Helps Data Mining Help MDE

When applying Lavoisier and Pinset to the software engineering field, we discovered a novel research trend where data mining techniques are being applied to modeling artefacts. In this trend, software models are analysed for different purposes. For instance, Basciani et al. [14] apply clustering techniques to large metamodel repositories to find groups with similar characteristics and for the automatic categorization and organization of these repositories. [9] also cluster metamodels by applying document retrieval and natural language processing mechanisms to, for instance, detect model clones.

As commented in Section 1.6.1, models in Model-Driven Engineering approaches conform to, or are instances of, a metamodel. These metamodels are often represented by means of object-oriented models, such as it happens when using EMF's Ecore notation (Steinberg et al. [152]). Therefore, a model can be also seen as data conforming to an object-oriented model. Consequently, we could analyse these data to study the different properties of a model.

As in any data mining context, FLANDM, Lavoisier and Pinset can be of help here. More specifically, Lavoisier and Pinset could be used in this context to construct datasets from models that could later be analysed with data mining tools and libraries. In this context, the use of Pinset is even more obvious, since EOL expressions are probably familiar to a wide range of MDE users.

To explore this hypothesis, we performed some dataset extractions in a case study inspired by the work of H. Osman et al. [68]. This work evaluates how different automatic feature selection techniques affect the accuracy of bug predictors. A *bug predictor* is a tool that uses a data mining process to determine whether a piece of code might be a potential source of bugs, usually within some confidence range (we introduced this analysis example in Section 1.2.3).

As input for the bug predictors, Osman et al. rely on an external dataset, provided by D'Ambros et al. [40], which has been widely used in the bug prediction literature (C. Couto et al. [27], Tantithamthavorn et al. [156]). This dataset collects data at the class level, by analysing source code hosted in software repositories augmented with some change metrics coming from other sources, such as configuration management systems. Therefore, the main entities under analysis are classes. For each class, a set of metrics, such as the ones defined by Chidamber and Kemerer [33], are calculated. Table 5.2 lists some of these metrics.

Table 5.2 Object-Oriented (OO) and Chidamber and Kemerer (CK) [33] class metrics.

Metric	Description
CK_WMC	Weighted method count
CK_DIT	Depth position on the inheritance tree
CK_NOC	Number of children
CK_CBO	Coupling between objects
OO_FanIn	Number of other classes that reference the class
OO_FanOut	Number of other classes referenced by the class
OO_NOF	Number of features
OO_NOA	Number of attributes
OO_NOPA	Number of public attributes
OO_NOPRA	Number of private attributes
OO_NOIA	Number of inherited attributes
OO_NOM	Number of methods
OO_NOPM	Number of public methods
OO_NOPRM	Number of private methods
OO_NOIM	Number of inherited methods

By using Pinset, the class-level metrics of Table 5.2 could be extracted from UML Class Diagrams, so that this kind of diagrams can also be used as data sources when training a bug predictor.

5.6 Evaluation

This section comprises the different activities we performed to evaluate the Pinset language. We started by determining if Pinset allows overcoming the Lavoisier limitations presented in Section 5.2.2, which were the main motivation for defining this new language.

Secondly, we considered comparing Pinset against some data transformation tools, such as SQL and Pandas, as we did in the case of Lavoisier. In the end, we decided to not perform this comparison, as the same results would have been obtained due to the similarities between Lavoisier and Pinset. Nevertheless, when exploring how data mining can help MDE (Section 5.5), we discovered that model-driven engineers sometimes used model transformation or model management languages, such as OCL (Object Management Group [124]), ATL (Jouault et al. [83]) or Epsilon's EOL (Kolovos et al. [98]) or ETL (Kolovos et al. [99]), for dataset extraction tasks. Since Pinset is

implemented over the Epsilon Platform, we decided to compare Pinset with ETL, i.e., the model transformation language of the Epsilon suite.

Next sections present the different parts of this evaluation.

5.6.1 Overcoming of Lavoisier's Limitations

In Section 5.2.2, we enumerated a set of limitations manifested by Lavoisier when performing certain advanced column calculations. Here we discuss whether Pinset features allow getting over these limitations.

- ***L1*: Not instance-based datasets**. Lavoisier's dataset generation is rooted in a *main class* of the domain model, and each row of the dataset corresponds to data from an instance of such class. As stated in Seccion 5.2.2, this method may prevent Lavoisier users from performing aggregations of several instances into the same row.

  Pinset avoids this problem through *typeless rules*. These special rules were described in Section 5.3.6, and allow defining datasets from a collection of elements of arbitrary type, not necessary belonging to the input model. As an example, Listing 5.10 shows some data aggregations performed at a year level.

- ***L2*: Arithmetic operations**. Lavoisier's syntax does not include support for typical arithmetic operations, such as additions and divisions. These operations might be required, for instance, to calculate derived values, such as ratios or differences.

  Pinset is based in the EOL language (Kolovos et al. [98]), which supports primitive types and operations between them. So, this kind of operations can be performed in Pinset if required. Listing 5.12 shows some of these operations performed over data from a Repository.

  Listing 5.12 Basic arithmetic operations over repositories data.

```
1 dataset operationsOverRepositories over r : Repository {
2   column name : r.name
3   column stars_and_watchers : r.stargazers + r.watchers
4   column stars_to_forks_ratio : r.stargazers / r.forks
5 }
```

- ***L3*: Support of advanced operators**. As Lavoisier was intended for users without programming skills, some advanced features such as first-order logic

operations were not included. However, these operations may be helpful for some users when manipulating data from a model.

Again, the use of EOL expressions in Pinset's syntax provides us with these advanced operations. Function *getUsedTechnologies* of Listing 5.9 (lines 1-7) is a good example of the use of first-order logic operations in Pinset scripts.

- ***L4*: Availability of imperative code structures**. For defining some columns, the use of imperative, structured code might be preferred, as the definition can be split into smaller steps that can be verified independently and more easily.

  In Pinset, we can define columns in blocks of EOL statements. For instance, the *star_to_forks_ratio* column of Listing 5.12 includes a division by the *forks* of a repository. If this number is zero, this expression would generate an arithmetical error. This error can be avoided by the use of an EOL block, where we first check whether the repository has any forks. If it does not, we return zero directly, to avoid the division error. Listing 5.13 shows the updated example.

  Listing 5.13 Use of a block to calculate a column through several statements.

```
1  dataset operationsOverRepositories over r : Repository {
2    ...
3    column stars_to_forks_ratio {
4      var res = 0;
5      if (r.forks <> 0) {
6        res = r.stargazers / r.forks;
7      }
8      return res;
9    }
10 }
```

- ***L5*: Reuse of intermediate calculations**. Lavoisier's syntax does not include that many features to perform calculations, so it is expected to also not offer a way to store intermediate values of such calculations. On the other hand, storing these intermediate values can translate into a performance improvement, more so if the values are used several times.

  Pinset allows to store these values in several ways: (1) when using an EOL block to calculate a column, we might define a variable through the *var* keyword, such as it is done in line 4 of Listing 5.13; (2) external functions in Pinset can be *cached*, so that each returned value of this functions is stored, and successive invocations of the same functions with the same parameters would return the

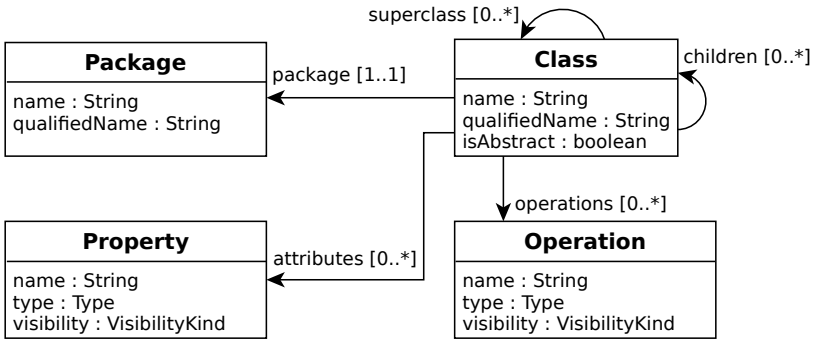


Fig. 5.4 Fragment of the UML Class Diagram metamodel.

stored value instead of calculating it again; and (3) Pinset offers the *NestedFrom* generator (see Section 5.3.5), which can be used to calculate a concrete value only once, and then it is shared by multiple column generators.

In summary, we believe that Pinset overcomes the Lavoisier limitations that we detected in some concrete contexts. Next section presents the case study that was used to compare Pinset with the ETL model transformation language.

5.6.2 Metrics Extraction with Pinset

We used the metrics extraction from class diagrams example introduced in Section 5.5 to compare the capabilities of Pinset against model transformation languages. Figure 5.4 shows a simplified fragment of the UML Class Diagram syntax that was queried when performing these extractions, expressed in an Ecore metamodel.

A *Class* has a name, and can be defined as abstract. Each class belongs to a *Package*. Both classes and packages have a qualified name, which includes the hierarchy of containers where they belong. For instance, if a class *Repository* is contained in a *github* package, which at the same time is contained in a *versionControl* package, the qualified name of this class would be *versionControl.github.User*. Classes can have *attributes* and *operations*. Attributes are *Properties* that, among other features, have a name, a visibility (e.g. public, protected, or private), and a type. *Operations* represent the methods of a class, and share the same features as the ones described for attributes, and others.

Now we show how Pinset can be used to extract the metrics contained in Table 5.2 from UML class diagrams. As a source of this kind of diagrams, we used the data gathered from open-source projects by R. Hebig et al. [136]. These data conforms to the metamodel of Figure 5.4.

Listing 5.14 shows how some of the metrics of Table 5.2 can be computed using Pinset. The dataset rule in this case is defined to use all *Class* type instances from the input model (line 1). Then, *Column* generators are used to extract the following information from each class: (1) its name (line 2); (2) whether it is abstract (line 3); (3) the name of its parent class, if any (lines 4-10); (4) its number of attributes (*OO_NOA*) (line 11); (5) its number of methods (*OO_NOM*) (line 12); (6) its number of features, i.e., the sum of attributes and methods(*OO_NOF*) (line 13); and (7) its depth inside an inheritance tree (*CK_DIT*) (line 14).

Listing 5.14 A Pinset dataset rule that extracts some metrics of Table 5.2.

```
1  dataset basicClassMetrics over class : Class {
2    column name : class.name
3    column isAbstract : class.isAbstract
4    column parentName {
5      var name = null;
6      if (not class.superClass.isEmpty()) {
7        name = class.superClass.first().name;
8      }
9      return name;
10   }
11   column OO_NOA : class.attributes.size()
12   column OO_NOM : class.operations.size()
13   column OO_NOF : OO_NOA + OO_NOM
14   column CK_DIT : class.dit()
15 }
16
17 operation Class dit(): Integer {
18   var dit = 0;
19   var node = self;
20   while (not node.superClass.isEmpty()) {
21     node = node.superClass.first();
22     dit += 1;
23   }
24   return dit;
25 }
```

As it can be seen, metrics from Table 5.2 can be computed with a relatively low effort by using Pinset. A Pinset dataset rule showing how to obtain all rules from this table can be found in Listing B.17 of Appendix B.8. Next section shows how the same metrics can be obtained with a model transformation language.

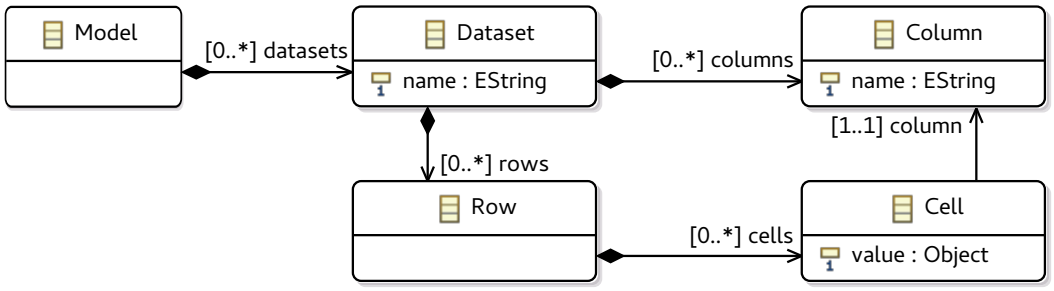


Fig. 5.5 Dataset Metamodel used as output in the M2M transformations.

5.6.3 Metrics Extraction with ETL

We show here how a transformation can be expressed in ETL to extract the class diagram metrics of our case study. ETL is a *Model-to-Model (M2M)* transformation language, so we need an output metamodel to serve as target of our transformations. ATL (Jouault et al. [83]), which is another M2M language, offers in its documentation a wide range of M2M transformation examples, including a specific entry for a table extraction scenario[4]. We used for our transformations the original table metamodel of this ATL transformation with slight modifications, in order to (1) allow the definition of several datasets in the same model; and (2) to explicitly store the column headers of each dataset, as it is usual in the data mining community. This metamodel is depicted in Figure 5.5.

A model conforming to this metamodel contains *Dataset* instances, i.e., one or more datasets. A *Dataset* is composed of a set of *Column* headers, plus a set of *rows*. Each *Row* stores *Cell* values, one for each column of the dataset. Each cell indicates to which column it corresponds.

The relationship between *Cell* and *Column* might be avoided by imposing an order both to the cells of each row and to the column headers. This way, cells in a certain position inside a row would correspond to the column header in the same position. Nevertheless, this solution makes instances and model transformations harder to maintain because of the required attention to ordering. For instance, if we removed a column of a dataset, we would need to update, in addition to the values of that column, how the values of subsequent columns are assigned, since these values would now correspond to a lower position. Keeping each cell associated to its column avoids this problem.

[4] https://www.eclipse.org/atl/atlTransformations/Java2Table/ExampleJavaSource2Table[v00.01].pdf

Listing 5.15 shows how the metrics extracted with Pinset in Listing 5.14 can be extracted with ETL. The ETL script computing all metrics of Table 5.2 can be found in Listing B.16 of Appendix B.8.

Listing 5.15 ETL transformation that extracts basic class metrics.

```
pre {
  var modelRoot = new Dataset!Model();
  var metricsDataset = createDataset("BasicMetrics");
  modelRoot.datasets.add(metricsDataset);
  var bmd_c_name = createColumn("name");
  var bmd_c_isAbstract = createColumn("isAbstract");
  var bmd_c_parentName = createColumn("parentName");
  var bmd_c_OO_NOA = createColumn("OO_NOA");
  var bmd_c_OO_NOM = createColumn("OO_NOM");
  var bmd_c_OO_NOF = createColumn("OO_NOF");
  var bmd_c_CK_DIT = createColumn("CK_DIT");
  metricsDataset.columns =
  Collection {bmd_c_name, bmd_c_isAbstract,
    bmd_c_parentName, bmd_c_OO_NOA,
    bmd_c_OO_NOM, bmd_c_OO_NOF,
    bmd_c_CK_DIT};
}

rule BasicMetricsClass2Row
transform class : Model!Class
to row : Dataset!Row {
  metricsDataset.rows.add(row);
  row.cells.add(createCell(bmd_c_name, class.name));
  row.cells.add(createCell(bmd_c_isAbstract,
                           class.isAbstract));
  var parentName = "";
  if (not class.superClass.isEmpty()) {
    parentName = class.superClass.first().name;
  }
  row.cells.add(createCell(bmd_c_parentName,
                           parentName));
  var OO_NOA = class.attributes.size();
  var OO_NOM = class.operations.size();
  var OO_NOF = OO_NOA + OO_NOM;
  row.cells.add(createCell(bmd_c_OO_NOA, OO_NOA));
  row.cells.add(createCell(bmd_c_OO_NOM, OO_NOM));
  row.cells.add(createCell(bmd_c_OO_NOF, OO_NOF));
  row.cells.add(createCell(bmd_c_CK_DIT, class.dit()));
```

```
39 }
40
41 operation Class dit(): Integer {
42   var dit = 0;
43   var node = self;
44   while (not node.superClass.isEmpty()) {
45     node = node.superClass.first();
46     dit += 1;
47   }
48   return dit;
49 }
50
51 operation createCell(col: Dataset!Column, val: Any): Dataset!Cell {
52   var cell = new Dataset!Cell();
53   cell.column = col;
54   cell.value = val;
55   return cell;
56 }
```

First, the dataset and its columns are instantiated in a *pre* block (lines 1-17). In ETL, a *pre* block contains code that is executed before any transformation rule. These blocks usually set up certain elements that need to be configured before running the transformations. In our case, the use of a *pre* block makes the defined datasets globally accessible during the execution process, which allows the transformation rules to populate them with rows. Thus, in lines 2-4 a new dataset called *BasicMetrics* is defined and added to the model root element. Then, in lines 5-16, the columns of the *BasicMetrics* dataset are defined and assigned. For the creation of datasets and columns, we rely on helper functions named accordingly (*createDataset*, *createColumn*). For the sake of simplicity, some helper functions have been left out of this listing, but they can be consulted in Appendix B.

It should be noticed that this *pre block* makes the dataset and the column headers globally accessible to any element in the transformation, avoiding that the transformation rules have to incorporate complex code to traverse the output model and recover these elements.

Once we have created the dataset schema, it is populated by the *BasicMetricsClass2Row* transformation rule. This rule transforms each class in a class diagram into a row in the output dataset (lines 20-21). This row is added to the *BasicMetrics* dataset (line 22), and then it is populated with a cell for each column. Thus, first of all, the name of the class is extracted (line 23) and assigned to the corresponding column header by means of the *createCell* helper function (lines 51-56). Then, in the same

way, the value for the *isAbstract* column is taken from the *isAbstract* class attribute (lines 24-25). Next, the *parentName* column value is computed (lines 26-31). To do it, we check if the class has a superclass. If so, the corresponding name is extracted from the parent class, otherwise this value is set to an empty string.

Other values, like *NOA*, *NOM* or *NOF* are obtained using the same techniques (lines 32-34). Finally, to calculate the *DIT* value for each class (line 38), due to its complexity, we have opted for extracting the code that computes it to an external function (lines 41-49).

We can see how the management of the dataset structure introduces extra verbosity and complexity in the extraction process. This verbosity obfuscates the final goal of the transformation, which is how elements from the input model get transformed into rows of the resulting datasets. We tried to alleviate this obfuscation by defining some helper functions and providing several global variables. This kind of variables, as everybody knows, might lead to undesired side effects.

It is worth mentioning that the described problems are not due to the use of a transformation language. If we have opted for using a general-purpose programming language such as Java, the problem would be worse, since this language does not offer facilities for manipulating models.

As conclusion, it can be stated that Pinset provides a higher-level, more compact, and less verbose syntax for computing datasets of model metrics as compared to languages such as ETL. Next section analyses this statement more in-depth.

5.6.4 Pinset vs. ETL Comparison

To make a more exhaustive comparison between Pinset and ETL-like languages, we devised several metrics computation scenarios. Each scenario makes use of different Pinset features, such as *Grid* or *NestedBlock* generators, to identify how the same functionality would be provided by a general-purpose model transformation language. The complete set of scenarios is described in Table 5.3, and the corresponding Pinset and ETL extraction scripts can be found in Appendix B and in an external repository[5].

We evaluated first the compactness of both languages by measuring the size of their scripts. For these measurements, we used the size in characters of the transformations, omitting any white space. For ETL scripts, some artefacts that are reused across scripts were not considered in these measurements. These artefacts are: (1) the helper functions for dataset management (Listing 5.15); (2) the dataset metamodel definition

[5] https://www.github.com/alfonsodelavega/pinset-examples

Table 5.3 Metrics extraction scenarios for the comparison.

Scenario	Description
S1: Basic Metrics	Get a set of class metrics as described in Section 5.6.2 and Listings 5.14-5.15.
S2: Features Accessors (Section 5.3.2)	Get the name, package name, and abstractness of a class.
S3: Extended Accessors	Get the name of a class, if it is abstract, whether it is a leaf (i.e. no subclasses), its qualified name, its visibility, and the name and qualified name of its package.
S4: Row Filtering (Section 5.3.3)	Process only those classes that are abstract.
S5: Grid (Section 5.3.4)	Get the number of attributes of a class grouped by visibility.
S6: Nested From (Section 5.3.5)	Get the name of a class, the name of its package, and the number of classes contained in that package.
S7: Typeless rules (Section 5.3.6)	Get the number of classes that meet certain thresholds for different metrics: number of attributes (NOA), number of methods (NOM), FanIn and FanOut.
S8: All Metrics	Extract all metrics from Table 5.2

(Figure 5.5); and, (3) the model-to-text transformations that would generate the final CSV files.

Table 5.4 summarises the results of this comparison. These results show that Pinset was able to reduce scripts size by more than two-thirds ($\backsim$69%), when compared with ETL scripts. This reduction is due to the use of high-level column generators specifically designed for certain data acquisition tasks. These primitives avoid the need to explicitly manage column creation. Moreover, some columns generators, such as feature accessors (see section 5.3.2), greatly help reduce script size. This specific syntactic sugar constructs are specific for dataset creation and, obviously, they are not avialable in model transformation languages, nor they will be. So, Pinset provides benefits for dataset creation, as it is specifically designed for this task.

Moreover, apart from a better compactness, Pinset provides some specific features that help simplify code, such as the management of null values. For some generators, when a reference is accessed, we do not need to check if this reference points to null. If it does, instead of raising an exception, Pinset provides an appropriate default behaviour, which most of the time prevents developers from having to take care of this issue.

The high-level syntax of Pinset also contributes to improve maintainability. Thanks to this syntax, we do not need to manage explicitly the dataset structure, which makes

Table 5.4 Size in characters/bytes of Pinset and ETL scripts.

Extraction Script	ETL/Bytes	Pinset/Bytes	Reduction
S1: Basic Metrics	1284	502	60,90%
S2: Features Accessors	744	96	87,10%
S3: Extended Accessors	1397	150	89,26%
S4: Row Filtering	350	90	74,29%
S5: Grid	892	287	67,83%
S6: Nested From	816	199	75,61%
S7: Typeless rules	1108	488	55,96%
S8: All Metrics	2220	916	58,74%
Aggregate of all scripts	8811	2728	69,04%

dataset definitions easier to maintain. As an example, if we wanted to include or remove a column from a dataset in Pinset, we would only need to update one section of the script, that is, the generator where the column is defined and calculated. In the ETL scripts, such as the one shown in Listing 5.15, there are three different places that would require modifications: (1) the section where columns are created (lines 5-11); (2) the statement where columns are assigned to the dataset (lines 12-16); and (3) the piece of code that calculates the values for that column (lines 23-38). This redundancy makes maintenance more complex, and it might lead to inconsistencies and errors.

These improvements in conciseness and maintainability seem to indicate that Pinset scripts might be easier to understand when compared to equivalent versions written in a generic model-transformation language. However, to be rigorous, this assessment needs to be confirmed with empirical experiments, where potential end-users, i.e., software engineers, evaluate Pinset against the tools they employ daily. This will be part of our future work.

Next section recapitulates the presented contents and concludes this chapter.

5.7 Chapter Summary

This chapter has presented Pinset, an domain-specific language for the extraction of two-dimensional datasets from models.

The creation of Pinset was motivated by some shortcomings detected when trying to use Lavoisier for calculating complex derived values, such as data aggregations. Pinset offers programming language constructs and conventional model-management operations, which ease defining these advanced calculations and should make this

language feel familiar to experts from the MDE field and, more generally, to users with programming experience.

When compared with existing model transformation tools, Pinset offers an equally powerful but more concise way of declaring datasets. This mainly happens because existing tools require preparing and managing the structure of the datasets explicitly while, in the case of Pinset, this structure is managed internally by the language. Therefore, it allows forgetting about boilerplate code and focusing on the features we wish to extract from the models. In addition, Pinset offers high-level constructs that facilitate the definition of dataset columns and the execution of typical dataset-related tasks that would need to be manually performed instead, such as normalisations and filling null values.

The existence of Pinset eases the inclusion of models as data sources in those data mining processes that aim to study data extracted from software projects. In addition, Pinset also facilitates the assessment of these models through advanced quality analysis techniques.

As future work, we will carry out more detailed performance and end-user tests, to empirically assess the language and to see what kind of new features would be well-received.

Chapter 6

Evaluation

6.1 Introduction

This chapter aims to assess whether the contributions of this thesis improve the state of the art and, as a result, might be helpful in real contexts. Once the different languages that comprise our work were defined, and the tooling that supports it reached an acceptable level of completeness and stability, we started a validation process to assess whether they could be used by average decision makers.

We describe here the different steps we performed to evaluate our DMDLs, which were the first focus of our evaluation. Precisely, these steps aimed to accomplish the following objectives:

1. Determine whether domain experts without knowledge in data mining techniques are able to understand and employ a DMDL to analyse bundles of data.
2. Identify any hidden problems in these DMDLs that might preclude its usage.
3. Collect feedback and suggestions for improvement to address in future works.

As the start of this evaluation, we compared our DMDLs with existing approaches in the data mining democratisation literature, to check whether our approach improves the state of the art. As a result of this comparison, we concluded that our approach provides the following extra benefits:

- A reduction of costs during the adaptation to specific contexts.
- Support for some analysis refinements.
- More facilities for dataset creation.

As a second step towards assessing whether average decision makers could make use of our DMDLs, we checked if these DMDLs were free of obvious usability flaws. For this purpose, we analysed whether our DMDLs fulfil a well-known set of usability heuristics (Nielsen [123]).

The analysis of these heuristics showed that, in general terms, our DMDLs offer different features that contribute to the satisfaction of those heuristics more directly related with the academic nature of our work.

To perform a more in-depth assessment of whether average decision makers are able of specifying and executing data mining tasks with our DMDLs, we performed a set of empirical experiments with different end users. We followed existing guidelines in software engineering experimentation (Barisic et al. [13], Wohlin et al. [170]) to plan and design these experiments.

In these experiments, we provided domain experts with a DMDL prototype generated with the FLANDM framework. This DMDL was devised for the educational domain, and the end users who took part in the experiments were university teachers from different areas, such as computer science, mathematics, and education.

During these experiments, participants had to solve some analysis tasks using our DMDL. These tasks were oriented to determine whether these participants were able to understand sentences written with our languages, as well as to produce them. The results showed that most participants managed to correctly complete these tasks.

As the reader can notice, these experiments did not yet attempt to assess the utility of *Lavoisier* or *Pinset*. Preparing the different artefacts that were necessary, coordinating the participants, executing the experiments, and processing the results required a considerable effort and time. Replicating these efforts for Lavoisier and Pinset would require around half a year, which was beyond the time frame for this thesis. Nevertheless, we will perform end-user tests with these languages in the near future.

The structure of this Chapter is as follows. Section 6.2 compares our DMDLs against state-of-the-art data mining democratisation approaches. In Section 6.3, we analyse how well DMDLs support well-known usability heuristics (Nielsen [123]). Then, Section 6.4 describes the planning and design of the performed end-user experiments. Section 6.5 presents and discusses the experiment results. Finally, in Section 6.6, any identified threat to the validity of these experiments is commented, and Section 6.7 recapitulates our findings and concludes the chapter.

Table 6.1 Coverage of the data mining process, including DMDLs.

Category\Stage Action	Collect	Preprocess	Mine	Interpret	Refine
Black-Box Components		✓	✓		
Development Frameworks	✓	✓	✓	✓	(✓)
DMDLs	✓	✓	✓	∼	✓

6.2 Comparison with State of the Art Approaches

In Chapter 2, we showed the results of our systematic review of the state-of-the-art of the data mining democratisation field. This review found a set of approaches that we organised in four categories: *workflows*, *Self-Service Business Intelligence (SSBI) solutions*, *black-box components*, and *development frameworks*. These categories and their approaches can be consulted in Table 2.9. In this section, we compare the features of our DMDLs against these approaches.

First, it is important to remember that our review concluded the following: (1) workflow applications are not ready to be used by average decision makers, as these applications are still to complex to configure and understand; (2) although SSBI solutions are prepared to be easy to use by decision makers without experience in data mining, this is only true for reporting and basic analytics tasks. In the light of these conclusions, we decided to exclude these categories from this comparison, and we focused on comparing DMDLs against black-box components and development frameworks.

In our systematic review, we defined an objective procedure to evaluate the approaches for the data mining democratisation field. We applied this procedure to our work. This procedure can be consulted in Section 2.1.6, and was composed of two parts: (1) determining the assistance offered by the approaches for the different data mining tasks; and (2) a set of quality attributes that are usually part of the different tradeoffs of performing an analysis process. The results of applying this procedure to our work can be found in Tables 6.1 and 6.2. These results are shown along with the ones for black-box components and development frameworks for easing their comparison.

With respect to assistance during the data mining process, DMDLs offer support for all stages, including the ability to refine such processes. For instance, users of a DMDL can change the data subset over which an analysis is performed by including a boolean comparison; or change the data columns under analysis by modifying some parameters of the available query commands. Most black-box components only support the data mining stage. While development frameworks also offer support for the whole

Table 6.2 Quality attributes by category, including DMDLs.

Quality Attribute	Black-Box Components	Development Frameworks	FLANDM DMDLs
Adoption Cost			
EQ2.1. Can it be deployed without adaptations?	Yes	No	No
EQ2.2. Can non-experts perform the adaptations?	-	No	No
Accuracy			
EQ2.3. Can the analysis reach expert-level accuracy?	No	Yes	Yes
EQ2.4. Can the approach be tuned to improve accuracy?	No	No	Yes
EQ2.5. Can non-experts perform the tunings?	-	-	Some
Completeness			
EQ2.6. What analysis techniques are available?	Predictive	All	All
Evolvability			
EQ2.7. How easy it is to extend the approach?	Hard	Med.	Med-Low
EQ2.8. Can new analysis techniques be included?	No	Yes	Yes

process, out of the 5 approaches in this category, only Santos et al. [145] provides some support for refining analysis processes. As a point to improve, our work did not focus that much in improving the interpretation of analysis results. We will study this stage in future research.

The results for the quality attributes reveal that, as expected, if we were to classify our work into one of the categories of the review, this category would be the development frameworks, as FLANDM offers languages that can be adapted to different contexts. Both these frameworks and DMDLs require some work to be deployed in a concrete domain, being a data scientist necessary to perform this work. In addition, both can reach expert level accuracy, as their analysis processes are customised for the characteristics of the application domain. On the other hand, one of the advantages offered by DMDLs over development frameworks is a greater support for some customisations or tunings of the invoked analysis processes. As these customisations are performed through small changes in the queries, they can even be performed by end users of these languages, e.g., decision makers. However, these customisations do not yet include the configuration of low-level parameters, such as controlling the concrete statistic employed in a correlation analysis. We will try to steadily include some of these parameters in future DMDLs to see if decision makers can make use of them. The rest of the quality attributes are the same: both approaches can use any data mining techniques, and they can be extended to support new analysis techniques. The FLANDM framework was developed to reduce development and maintenance costs, so we might also benefit from a lower effort when extending or updating a DMDL.

For the quality attributes comparison with black-box components, these have the advantage of not requiring any changes to be used in a new domain. However, as previously stated, the lack of adaptations of these components to the peculiarities of the concrete domain might hamper the accuracy of the analysis results. This possible loss of accuracy would not affect development frameworks or DMDLs, as these approaches can be adapted to avoid this problem.

In summary, when attending the different tradeoffs of each analysis context, DMDLs seem to offer advantages over the other approaches for data mining democratisation available in the literature. Next sections aim to determine if these languages can be directly used by average decision makers.

Table 6.3 Usability heuristics defined by Molich and Nielsen [117].

#	Heuristic
H01	Simple and natural dialogue.
H02	Speak the user's language.
H03	Minimise the user's memory load.
H04	Consistency.
H05	Feedback.
H06	Clearly marked exits.
H07	Shortcuts.
H08	Good error messages.
H09	Prevent errors.
H10	Help and documentation.

6.3 Fulfilment of General Usability Heuristics

As one of the first steps for the evaluation of our languages, we checked whether these languages were free of obvious usability flaws. During the development of the FLANDM framework, we focused on making the generated DMDLs as usable as possible. For this reason, these DMDLs have a simple and easy to use syntax, and several assistance features, e.g., autocompletion and validation mechanisms (see Sections 3.3.5 and 3.3.3 respectively). Therefore, we expected our languages to be free of the mentioned flaws.

To confirm this hypothesis, we studied how well our DMDLs supported a set of general *Usability Heuristics* (Gerhardt-Powals [65], Holcomb and Tharp [75], Nielsen [123]). Usability heuristics can be understood as sets of principles to guide the creation or evaluation of end-user software interfaces. Although heuristics analysis is not as good as empirical experiments, it is an inexpensive method to perform system assessment, and it serves as a starting point that can help focus future user testing evaluations.

For this analysis, we used the set of ten heuristics defined by Molich and Nielsen [117], who enumerated general considerations that can be applied to most types of interfaces, from character to graphical-based ones. These heuristics can be found in Table 6.3.

Some of these heuristics have an industrial viewpoint, in the sense that they focus on complying with some requirements that are more typical for production-ready software. It should be remembered that we are evaluating academic prototypes built as proof of concept for validating a scientific work. Therefore, these prototypes lack of certain features, such as help documentation, that should be found in a fully-developed system. Therefore, we discarded some heuristics that were not of interest for our objectives.

More specifically, we discarded the heuristics referring to providing continuous feedback to the end user of what is currently happening when navigating the windows and menus of the tool (*H05*); to clearly defining how to exit or cancel current processes (*H06*); or, as commented, to evaluating the quality of the provided help material (*H10*). These more pragmatic issues will be addressed in the future.

Here we describe the remaining 7 heuristics that we studied from the context of a DMDL language generated with FLANDM:

- **H01: Simple and Natural Language**. This heuristic refers to simplifying user interfaces as much as possible. Every element appearing in the interface has to be justified. Also, interfaces have to feel as natural as possible when performing the tasks for which they were designed.

  DMDLs' syntax has been made as concise and simple as possible, only requiring the command to apply and the data this command would process. This syntax follows the natural structure of giving simple orders, in a similar fashion as giving orders to another person.

- **H02: Speak the User's Language**. The dialogue between system and user should employ concepts and vocabulary from the user's domain, this is, it should be familiar terminology to end users.

  This is one of the main advantages of using DSLs in general: commands can be renamed/created to reflect domain-specific tasks, and the vocabulary that is used comes directly from the day-to-day terminology of the application domain.

- **H03: Minimise the user's memory load**. This heuristic relates to reducing the information that users have to mentally store when employing a system.

  When using a DMDL, users do not need to memorise the concrete syntax of commands or entities: DMDLs have a proposal provider that shows such elements in a list-like format to choose (see Section 3.3.5). Lastly, the fact that the DMDL commands and vocabulary are close to the domain (as commented in *H2*) might make the syntax easier to remember.

- **H04: Consistency**. The functionality of the system should provide a feeling of consistency: the same set of available actions should be offered throughout the different system windows.

  With respect to this, the syntax of the different commands of the DMDLs is very similar, as only some parameters of these commands may differ. This similarity

gives a homogeneous and consistent image to the end user, who only has to remember one general syntax structure.

- **H07: Shortcuts**. The interface should include some accelerators (e.g. keyboard shortcuts), so that experienced users in the systems can go faster than using the basic and more time-consuming processes that are available for non-experts to work with the systems, e.g., typical wizards that require end users to steadily navigate through several windows to perform a task.

  DMDLs provide an autocompletion mechanism to write queries based on the available commands and data (described in Section 3.3.5). Once a user gets familiar with this mechanism, queries can be written and tuned considerably faster, as users do not have to type by themselves the whole query.

- **H08: Good error messages**. Error messages should be as informative as possible, and they should be expressed in natural language. In addition, error messages should suggest ways to solve any encountered error.

  A validation mechanism is provided within the DMDL editor to assist users during the composition of queries (see Section 3.3.3). This system also uses terminology from the domain, so it should feel natural and helpful to the end users. For instance, if the user performs row filtering over an entity by an attribute that does not belong to such entity, a very precise message such as *"the entity* `entityName` *does not have an attribute named* `attributeName`*"* is reported. Similar messages are offered in other contexts. The suggestion of error fixing options is something that is still pending, and it could be a nice feature to include in a near future.

- **H09: Prevent errors**. Rather than providing a good error notification mechanism, the system should limit the number of errors that a user may commit as much as possible.

  The combination of autocompletion and validation mechanisms gives users good advice about how to compose the queries, so if users employ these mechanisms the errors they encounter should get reduced. In addition, the system does not allow users to execute a query that has validation errors until fixed, avoiding any posterior malfunctions during the analysis caused by, for instance, a badly written entity name, or an incompatible attribute comparison (e.g. comparing a number with a textual string, or using a numerical comparison like *moreThan* in a nominal attribute such as "gender").

To sum up our comments about usability heuristics, DMDLs generated with the FLANDM framework seem free of obvious usability flaws. The adaptation of the syntax to the concrete domain should make the generated DMDL easy and natural to use; and the validation and autocompletion mechanisms described in Sections 3.3.3 and 3.3.5 should help users reach the correct formulation of an analysis query.

However, as commented before, this heuristics evaluation is not enough to certify that, by using our languages, average decision makers would be able to execute data mining tasks. For that reason, we opted for validating these initially promising results by performing some empirical experiments. Details about how these experiments were carried out are given in the next section.

6.4 Empirical Experiments

We start this section by giving an overview of the main aspects of the performed experiments. After that, more concrete details of these experiments are provided.

6.4.1 Overview

Software engineering experiments usually follow a comparison-based approach (Wohlin et al. [170]). In this kind of approach, a new solution or method is evaluated against what is currently available in industrial practises, which are often referred as the *control* or *baseline* solutions. The comparison is done in a controlled environment, where most variables are fixed, and only the ones we are interested in measuring are affected during the experiment.

For instance, if we wanted to assess whether *Test-Driven Development (TDD)* methods (Beck [16]) provide any benefits when compared with traditional, tests-after-code methodologies, we could perform an experiment where some software programs are developed by two groups of participants: the first group would follow TDD methods, while the second one would use the traditional approach (i.e. the baseline). Then, we could compare the results to assess whether TDD improved the variables under analysis (e.g. code readability or number of bugs) with respect to the baseline approach. An experiment of these characteristics was performed and discussed by Santos et al. [144].

Nevertheless, in our case, there is no baseline practise against which we can compare our work. Using current practices, domain experts without knowledge in data mining are not able of executing data mining tasks by themselves. Therefore, we cannot follow a classical, comparison-based approach.

Table 6.4 Summary of the performed empirical experiments.

Criteria	Description
Domain	Analysis of Educational Data
DSL	Educational DMDL
Quality Focus	Understandability, Learning Curve
Context	Offline, Professional, Real Problem, Specific
Participants	University Professors (3 areas), Graduate (PhD) Students
#Participants	Computer Science (9), Mathematics (8), Education(7) Graduate Students (13)
Training	Previous explanation and demo training (∼30 min)
Experiment	Pre: multi-choice questionnaire on participants skills Evaluation: DMDL-based test Post: usability questionnaire
Runs	Single case study on 4 heterogeneous groups
Results	- Teachers were able to understand and use the DMDL to launch data mining processes. - Most teachers answered all questions correctly. - Teachers's opinion on DMDL's usability was positive.

Consequently, to analyse whether our work helps these domain experts, we used a different approach: we designed a set of analysis tasks to be completed by employing one of our DMDLs. If domain experts were able to achieve these tasks, it would be an indicator that our DMDLs satisfy our initial goals.

We applied the previous approach in a set of experiments devised around a DMDL for the educational domain. The experiments were planned and designed with the help of the software experimentation guidelines promoted by Wohlin et al. [170] and Barisic et al. [13]. According to these guidelines, we first established the scope and goals of these experiments. Then, we analysed potential participants, selected the most suited ones, and designed the experiments.

Before carrying out the designed experiments, a pilot test was performed to assess the correctness of the different experiment components. The participants of this test had expertise in FLANDM concepts, but they had not previously used any prototype DMDLs of any kind. As a result of this pilot test, some issues were detected and fixed, e.g. the training presentation was improved, and some minor inconsistencies in the educational DMDL's syntax were fixed.

Table 6.4 gives a summary of the main aspects that conform the planning and design of the performed experiments. Next sections give more details on these aspects.

6.4.2 Scope

Before planning and executing any experiment, it is important to determine its goals, to ensure that the results of this experiment will be useful for the purposes it was defined.

We established the scope and objectives of our experiments via the *Goal/Question/Metric (GQM)* paradigm from Basili and Rombach [15]. In our case, the goal was to determine whether domain experts, without knowledge on data mining, are able to execute data mining tasks using our languages. For this goal, the question is trivial: *How well can domain experts execute data mining tasks with our languages?*. To measure this question, we proposed domain experts to complete some analysis tasks of increasing complexity, and we gathered how many of these tasks they were able to complete.

To perform these experiments, we selected some participants for the role of domain experts without knowledge in data mining. As we were surrounded by university teachers, and these teachers lack this knowledge, we decided to select them as participants. Consequently, the DMDL that these experts used during the experiment targeted the educational domain, in the university context. In summary, in our experiments, university teachers tried to complete some analysis tasks over data from a university course with the help of an educational DMDL.

With this information, and following the GDM directives of Basili and Rombach [15], the scope of these experiments was established by filling a predefined *template*, where the following aspects are specified: the *objects* under study; the *purpose* of the experiment; the *quality focus* of the experiment; the *perspective* or point of view from which the experiment is carried out; and the *context* of the experiment, this is, determining the concrete *subjects* (participants) and the *objects* (software artefacts) that intervene in the experiment.

Each of the above aspects is defined in the GQM template, one aspect per line. For the carried out experiments, the GQM template would be as follows:

Analyse < *DMDLs* >
for the purpose of < *enabling domain experts to execute data mining tasks by themselves* >
with respect to its < *understandability and learning curve* >
from the point of view of < *any domain expert* >
in the context of < *university teachers analysing data from a course* >.

Next section details the context of the experiments we performed to accomplish the goals of the above template.

6.4.3 Context

After setting the scope of our experiment, the next step involved determining its context. For that purpose, the following four context dimensions were determined:

1. **Online vs. Offline**. *Online* experiments are performed in real, production environments, using professionals or clients as subjects. For instance, Amazon performs *A/B Testing* on its e-commerce webpage by offering different views to its clients, and then comparing the performance of these views, e.g., by analysing which view was more successful in selling products) (Hill et al. [73]). *Offline* experiments are executed in a controlled environment that is separated from the real one, but that tries to resemble it as much as possible.

   In our case, an online context would have required to gather data from the courses of each one of the participants. This was not an affordable task, so we opted for executing an offline experiment where all participants worked with data from the same course.

2. **Student vs. Professional**. It is hard to find companies willing to lend their workers' time to perform empirical experiments. So, *students* are commonly employed in these tests instead of the real *professionals*, although this change might imply that the experiment results are less reliable (Falessi et al. [57]).

   For our experiments, fortunately, we could use university teachers, who are target professionals of an educational DMDL. In addition, in order to increase the number of subjects, a set of PhD students also participated in the experiments. Nevertheless, most of these students had some previous experience as teaching assistants, so they qualify as quasi-professional subjects for our empirical experiments.

3. **Prefabricated vs. Real Problems**. The use case of an experiment could be *prefabricated* for that concrete setup, or it could come from existing, *real* projects.

   We used course datasets that follow the structure of what can be found in professional e-learning platforms, such as Moodle (Rice [138], see also Section 1.2.1). Consequently, these datasets are the typical ones used in educational data mining research (Jugo et al. [84], Romero and Ventura [140], Zorrilla and García-Saiz

[176]). However, for data protection reasons, we were not allowed to use the real data of these datasets in the experiments. So, we had to replace this real data with synthetic one. We introduced specific patterns in the synthetic data to be found by the participants when invoking analysis processes with the provided DMDL. In any case, the structure and nature of the employed data were the same as if these data were real. Therefore, we are inclined to say that our experiments' setup resembles more of a real problem than of a prefabricated one.

4. **General vs. Specific**. The conclusions of an experiment could be generally applicable to a whole field, e.g., Software Testing; or to a specific subfield, such as Software Testing for Embedded Systems.

   The results of these experiments would be exclusive of the educational domain. These results might give some clues about how experts from other domains would behave when using a DMDL from their domain. Nevertheless, from the results of these experiments, we cannot make any sound statement about other domains. This will be a subject of future work.

In summary, based on the reasons presented above, the context dimensions of our experiments are *offline*, *professional*, *real problem* and *specific*. Next section describes the subjects that participated in these experiments.

6.4.4 Participants Selection

To represent a real test for the applicability of the DMDL, participants would ideally have limited or no knowledge at all in data mining techniques. As mentioned in the previous sections, the context of these experiments was the educational domain. We managed to enroll a total of 37 subjects in our experiments. These subjects can be organised in the following two sets:

1. **University teachers**. The educational DMDL was tested by 3 groups of university teachers, each of them belonging to a different area. Namely, the experiments included 9 teachers of Computer Science, 8 of Mathematics, and 7 from Education, making a total of 24 university teachers. From these, only 2 Mathematics teachers declared to have extensive experience in data mining techniques.

2. **Graduate Students**. A fourth group of 13 students, mostly PhD candidates, accepted to participate in the experiments. A good characteristic of this set is that

Table 6.5 Available commands in the improved educational DMDL.

Command	Description
find_reasons	Searchs for causes of a concrete data phenomenon.
show_groups	Groups data according to simmilarities.
describe_entity	Shows a description of an entity and its attributes.
show_data	Shows the tabular data of a given entity.
attribute_ranking	Shows a ranking of the most-related attributes to a given one.
show_relation	Calculates the relation between two attributes of an entity.

it contains participants from a broad range of research topics, such as Mathematics, Physics, Computer Science, and Civil Engineering. Additionally, most of these students have teaching experience in university subjects, so they are knowledgeable of the educational context that is used in the analysis. Although the degree of expertise in data analysis techniques varied between these participants, none of them worked directly in the data mining field, or considered herself a data mining expert.

In summary, these four groups of participants can be considered experts in the educational domain and, although some of them are proficient or have some knowledge in data mining, most of them have no skills in this field. This lack of these skills made them a good group of subjects to test our educational DMDL. At the same time, having some experts in the data mining field participating in the experiments could give us another professional perspective about how to improve our DMDLs.

Next section describes the educational DMDL prototype that these participants used in the experiment.

6.4.5 DMDL Prototype

This DMDL is an updated version of the original data mining DSL for the educational domain that started our work in data mining democratisation. We introduced this original DSL in Section 3.1. The changes performed to this updated DMDL with respect to the original are the following:

1. First, we ported the original DSL to the infrastructure defined by FLANDM.

2. One important change that we introduced in this DMDL was that, to better test the language in the local context of the participants, we translated its syntax

to Spanish. Although most of the selected teachers were English speakers, we considered that it would be easier for them to learn a new technical language in their mother tongue, than having to make an extra effort to understand the presented concepts in the English language. Additionally, it is important to highlight that, for a subset of the participants in this experiment, this DMDL was the first contact they had with a programming language of any kind. So, again, establishing this first contact in the mother tongue of the teachers seemed an easier endeavour. The modular structure of FLANDM allowed us to translate the grammar, validation and result messages with reduced effort (see Section 3.5.3).

3. We increased the number of available commands from the original two types of queries available in our first prototype, so that we had a wider range of options to propose analysis tasks. The complete list of commands of the updated DMDL is shown in Table 6.5. Consequently, new analysis subprocesses were also included among the available ones in FLANDM.

4. Lastly, for the data to be analysed with this DMDL, we included three entities that stored students information of different nature:

   - Demographic data, which included, among others, whether the student was repeating the course, class assistance, or if Spanish was her mother tongue.
   - Activity data, as gathered from the e-learning platform (see Section 1.2.1). This data included, for instance, the number of sessions in the platform, the average duration of these sessions, or the usage of the platform's forum, e.g., number of visits and written messages.
   - A third entity containing both demographic and activity data, which was useful to launch analyses over the complete set of students information.

   As stated in Section 6.4.3, we were not allowed to use the real data of these datasets in the experiments because of data protection regulations. Instead, we maintained the structure of the datasets, i.e., the existing columns, with respect to the original ones, and then filled these datasets with synthetic data. So, although not with the original data, participants used the educational DMDL in the same manner and over the same data bundles as they would if the data were the original one.

Once the infrastructure for the experiments was ready, we designed the set of tasks that participants would have to complete, and the different measurements that would

Table 6.6 Statements of the Pre questionnaire.

Task	Statement
Basic Office	I use basic office apps such as word and presentation processors.
Basic Calc Sheets	I use calc sheets just to tabulate data.
Advanced Calc Sheets	I use advanced calc sheets functions such as formulas and graphs to analyse data.
SQL	I use the SQL language to query and manage data in a relational database.
Analysis Tools	I use GUI-based analysis tools for the analysis of data, e.g., Tableau, Rapidminer, IBM SPSS.
Analysis Libraries	I use libraries or programming languages focused in the management or analysis of data: R, Pandas, Scikit-Learn, Weka, or similar.

be taken during the experiments. We divided these experiments in three parts: a pre-test, a test, and a post-test. The pre-test aimed to know the degree of familiarity with data mining techniques of the participants. The test contained the experiment itself. The objective of the post-test was to collect participants' opinions about the provided language. These parts are described in the following sections.

6.4.6 Pre-Test: Assessment of Skills

A pre-test was filled by the participants at the beginning of each experiment, with the objective of determining their expertise level in data management and/or analysis techniques. To achieve this objective, we gathered information about the participants' usage of these techniques with the following questions:

- First, we asked participants whether they employ data management and data analysis techniques during their daily activities. In addition, we included a free text field to allow them to briefly describe these activities.
- Second, we showed a set of statements about different data-related tools that participants may use in their activities. Table 6.6 shows an English translation of these statements. The statements were showed in increasing difficulty, i.e., we started with tasks related to office apps and spreadsheets, and finished with the use of data mining libraries. Users had to select those that matched their activities. Additionally, they could also include any extra tool that was not present in the statements by using a free-text field of the form.

6.4.7 Test: Execution of Data Mining Tasks

The second section of the experiment involved the proper test, where participants had to complete a set of data mining tasks or challenges. For each challenge, the participant had to provide an answer for a question with the help of the educational DMDL. These questions are shown in Table 6.7.

The set of test questions was designed to steadily increase the complexity of the tasks. These increments had the objective of offering a steady test, so that participants would not feel frustrated of not knowing how to solve the first questions; and to determine how far these participants could reach in understanding and using the DMDL.

For instance, *Q1* only involved indicating which command was the appropriate one to perform a task, while in *Q2* participants had to describe what was the objective of a given query. This is, *Q2* aims to determine whether domain experts are at least able to understand the language. In *Q3*, participants had to copy and execute a given query into the environment, and then interpret the results. Question *Q4* involved refining the introduced query in *Q3*. In *Q5*, participants received a query that contained mistakes to fix. *Q6* and *Q7* required to write a query from scratch, and then refining it, respectively. Finally, the last question (*Q8*) required to write a complete query from scratch and from an informal description of the objective of the analysis, so participants had to understand the requirements and compose a concrete query to get the results.

In addition, some questions had a secondary objective of testing certain features of the DMDL, e.g., the validation mechanism (*Q5*), or the assisting commands, such as the **`describe_columns`** command (*Q8*).

For each question, participants had to give an answer in an online questionnaire. In addition, participants were asked to copy the DMDL query that led them to the provided answer, so that any mistakes could be matched with the formulated query. The original set of questions in Spanish that participants answered in the test can be found in Appendix D.

6.4.8 Post-Test: Satisfaction Questionnaire

After completing the data mining tasks of the test, participants had to answer another questionnaire, where we asked for their opinion about the general usefulness of the language. This questionaire was composed of a set of statements, which participants have to grade within a Likert scale from 1 to 5, being 1 a strong disagreement with the statement and 5 a strong agreement. This set of statements is shown in Table 6.8.

Table 6.7 Translated test questions corresponding to data mining tasks.

#	Question
Q1	If we want to compare two columns of a table to determine if there is any relation between them, ¿which command should we use?
Q2	What is the objective of the following query? *<**show_data of** students_demographic **with** origin **equals** highschool>*. Describe it briefly.
Q3.1	Execute the query *<**show_groups of** students_activity **with** origin **equals** highschool>*, whose objective is to look for the different student profiles that exist in the course based on the student activity. How many groups or profiles appear?
Q3.2	What is the average total sessions number for group 1?
Q4	Modify the previous question, so that now only those students that failed the course are grouped. What is the new number of groups after including this modification?
Q5	The next query searchs for reasons why students have failed the course, using for that task all the available data: *<**find_causes of** course_result **equals** 0 **of** all_students_data>*. However, this query contains mistakes. Copy the query into the editor and try to fix these mistakes.
Q6.1	Write a query that, using the activity data, shows the more related columns to the course outcome of the students. What is the most-related activity column?
Q6.2	What is the second most-related activity column?
Q7	Repeat the previous query, but now use the demographic data. What is now the most-related column with the course outcome?
Q8	We want to study whether there is a relation between two activity columns: total number of sessions in moodle, and number of written messages in the forum. Write a query that obtains this relation and introduce here the numerical value of this relation. You can use the **`describe_columns`** command to discover the concrete column names you should use when writing this query.

Table 6.8 Post questionnaire.

#	Statement
Post1	Commands of the DMDL were easy to understand.
Post2	I considered easy to interpret the objective of a given query.
Post3	The use of vocabulary from the educational domain was useful to better understand the objective of an analysis.
Post4	Error messages provided by the validation system were useful to identify and fix query mistakes.
Post5	I considered easy to include modifications in previously written queries.
Post6	The autocompletion system was of help when creating or updating queries.
Post7	I considered easy to create queries from scratch.
Post8	I think that users without knowledge in data mining techniques might find this kind of languages too complex to use.
Post9	I consider necessary to receive a training session for the correct usage of this language.
Post10	In general, I think this language is easy to use.
Post11	I would consider positive/useful the existence of a language of these characteristics in my working domain to help me with daily tasks.

For almost all statements, a strong agreement indicates a positive opinion. Only one statement, *Post8*, was the opposite: a strong disagreement represents a positive outcome. This negated statement was introduced to help determine the attention level with which participants were answering this questionnaire. Along with their Likert-scale points, participants could also include a rationale for the given grade.

These statements follow the same incremental complexity as included in the tests. Moreover, some concrete statements aimed to gather the opinion that participants had on some features of the DMDL, such as the validation (*Post5*) and autocompletion (*Post6*) systems.

In addition, participants were asked about their opinion concerning the easiness of use of the DMDL (*Posts 8 and 10*) and whether they would find useful to have a DMDL adapted to their domain to help them with their daily activities (*Post11*).

Lastly, a final free-text field was left for users to express any opinions on the DMDL or in the performed experiment.

6.4.9 DMDL Training

As it was never our objective for domain experts to be able to use a DMDL without any initial guidelines, we designed a training session that participants received before performing the experiments.

Some of these participants had zero knowledge about data mining techniques, so the training started with a seminar about how these techniques can be used to analyse data. In addition, the complexity problem associated with democratising data mining was also described, which allowed us to introduce our contributions to this area, including the DMDLs.

In the second part of the training, a demo of how to use a DMDL was shown. Participants followed a brief seminar, where we presented the syntax of a DMDL, its commands, and the Eclipse environment used during the experiment.

During this demo session and also in the test, printed documentation in the form of a two-page manual was provided. Participants also had this manual available during the experiments execution. The manual included, on one page, a summary of the DMDL's query syntax, and the available commands along with a small description of how to employ them with their parameters. The second page was a summary of the entities that could be analysed with the DMDL they were using. The manual can be found in Appendix C.

The syntax and the available commands of the DMDL employed in this demo session were the same as in the educational DMDL used for the experiment. Nevertheless, to avoid influences and biases in the experiments, the domain of the training DMDL demo was finance, instead of education. During this demo, participants were guided in the analysis of data from bank clients, with the objective of focusing phone marketing campaigns in those clients that a priori seemed more keen to accept a term deposit offer. These data included different indicators, such as the client's balance, whether the client had a credit or mortgage, among others. Along with these data, the result of previous marketing calls was also present, which allows, for instance, to train a prediction model to identify those clients more inclined to accept a future marketing offer.

Apart from the domain change, and again to avoid bias, training and experiment sessions were separated in time. This separation had the objective of disconnecting subjects' minds between the sessions, so that the test did not simply involve copying and adapting the queries they had just seen in the training session. As a minimum, two hours passed between the training and test sessions, being this interval in the order of days for half the experiments.

We finished this training by giving participants an overview of the test structure, so that in the test session only a recap of the most important details was necessary. Next section discusses the obtained results in these experiments.

6.5 Analysis of Results

Here we show the results obtained for the three sections of the experiment, in the same order that the participants performed them.

6.5.1 Pre-Test

The first question of the Pre questionnaire asked participants whether they tried to gather and analyse data in their research and/or teaching activities. Their answers show that only ∼40% of the participants perform these data-related tasks. Most of them do it to see if there is any link between the performance of their students with other indicators of the course, such as the use of the learning materials, the assistance to classes, or even the students' opinion gathered at the end of the course.

In some cases, participants indicated they perform analyses for research purposes, e.g., determining the distribution of a random number generator, performing statistics of laboratory data, or calculating psychological indicators.

On a side note, some participants also mentioned that they do analyse data for personal finance purposes, or to determine how their time is distributed among the different activities of a day.

Figure 6.1 shows the analysis tools usage answers of the different groups. We can see that most partipants employ basic office and spreadsheet programs, with many them also using advanced spreadsheet functions. Interestingly, none of the participants use the SQL language in their daily work. Another interesting detail is that, while most university teachers from the Education group are familiar with GUI-based analysis tools, such as IBM SPSS[1], this kind of tools is not very popular for the other groups. Lastly, only some participants from the Mathematics group and a small part of the Grad students have some experience with advanced data mining libraries or languages.

Apart from the tools of Figure 6.1, participants could also indicate any other tool they use in a free field of the questionnaire. Different tools were indicated in this field:

- Reporting facilities of the Moodle e-learning platform, which allow, for instance, to study how the students use the available learning materials.

[1] https://www.ibm.com/analytics/spss-statistics-software

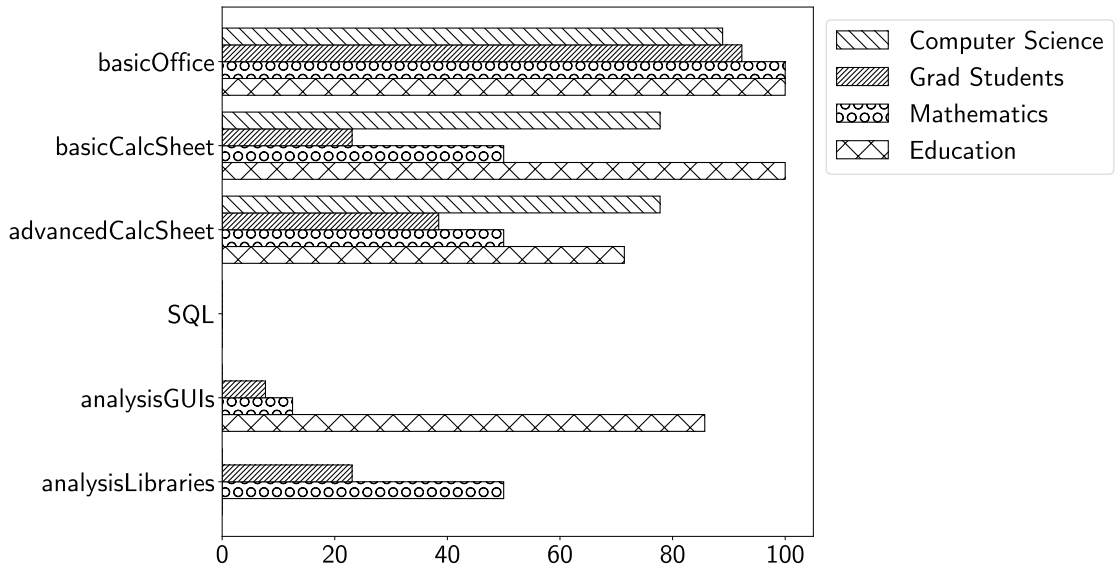


Fig. 6.1 Pre questionnaire answers of tools usage.

- Data analysis tools such as Kaleidagraph[2] or Minitab[3], for analysing research projects' data. These tools can be used to generate data graphs, including confidence or error intervals; or to perform basic statistics, such as Student t-tests, ANOVA, or Wilcoxon, among others.

- Software to perform qualitative analyses of natural language texts. The mentioned tools were QDA Miner[4] and ATLAS.ti[5]. These tools were used, for instance, to objectively measure the richness of the student delivered texts when performing argumentation assignments.

- Anecdotally, one participant mentioned that she tries to find links between students' course outcomes and other indicators *intuitively*, this is, by manually looking at the data and trying to find an evident relation without employing any tool.

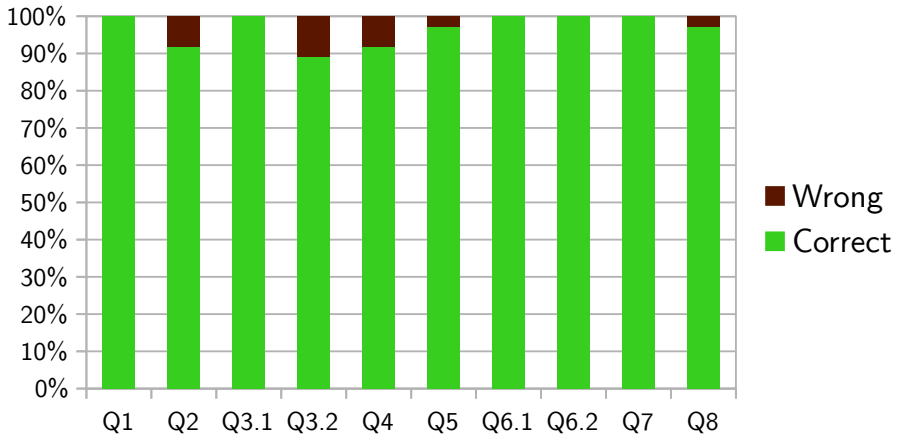


Fig. 6.2 Aggregated results of the test.

6.5.2 Test

In Figure 6.2, the aggregated results for all participants are shown. As it can be seen, participants completed their tasks correctly in most of the cases. So, it can be stated that teachers without experience in the data mining field were able to use our DMDL to execute analysis processes.

Figure 6.3 shows the results of this test for each one of the four groups. The following aspects can be highlighted:

- Disaggregating the results, we can se that these were still positive across all groups. In addition, the graduate students group was very multidisciplinary by itself. Therefore, we can conclude that the DMDL can be used by any teacher, independently of his area.
- A slightly higher number of mistakes can be found in the group formed by Education teachers. This was somewhat expected, as their background was the less close to computer technologies of all the groups. Nevertheless, this number of mistakes was not significant in general terms.
- From the answers, we detected that the statement of question *Q3.2* might have led to some misunderstandings on what was the correct answer. Question *Q3* asked participants to copy and execute a given grouping query (see Table 6.7

[2] http://www.synergy.com/wordpress_650164087/kaleidagraph/
[3] https://www.minitab.com/en-us/
[4] https://provalisresearch.com/products/qualitative-data-analysis-software/
[5] https://atlasti.com/

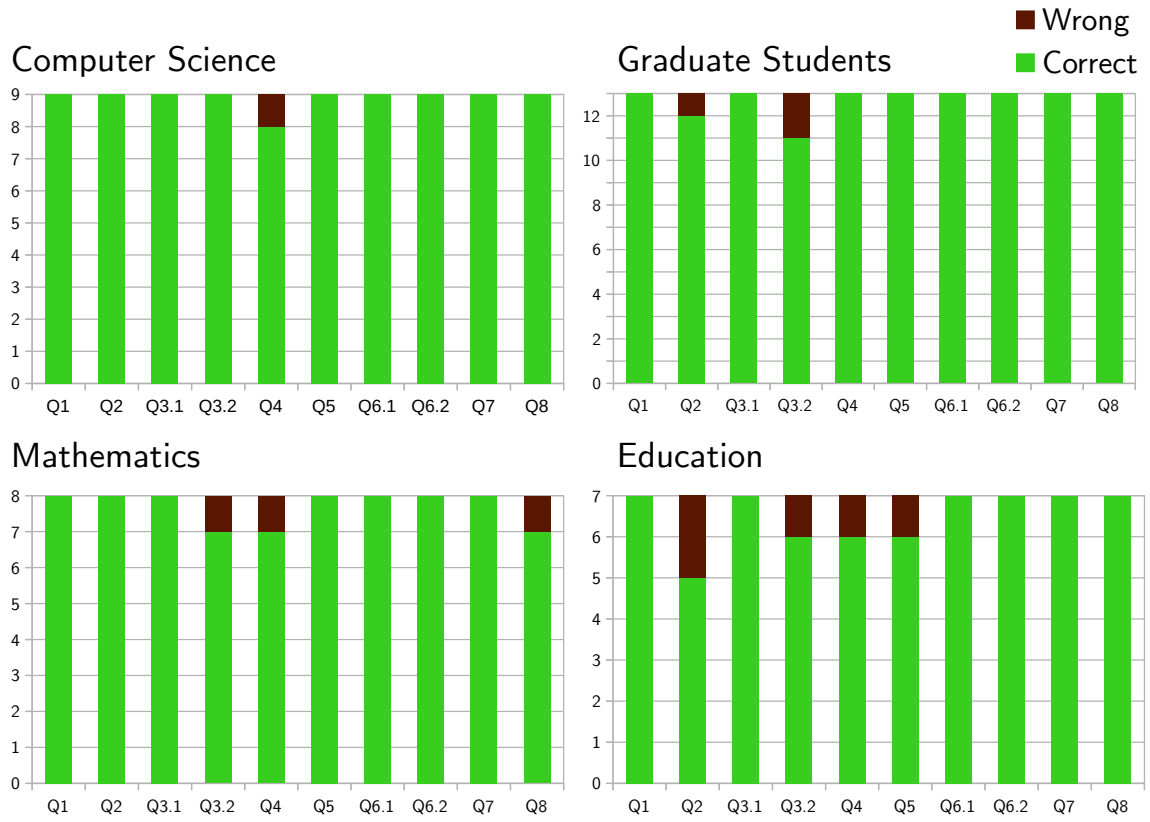


Fig. 6.3 Results of the test for the different groups.

or Appendix D). In sub-question *Q3.2*, a concrete numerical answer about the average total sessions of group 1 was required. Four participants answered incorrectly this question: one of them gave the value of another group (returned groups by the clustering algorithm were labeled starting from 0, so group 1 was misunderstood with being the first one); the other three answered the value of another row of the correct group: instead of giving the *average total sessions* value, they introduced the *average total duration of a session*. We will try to avoid this kind of misunderstandings in future experiments.

The mistakes made by participants, which were scarce, were the following:

- A bad explanation of the objective of the ***show_data*** query (*Q2*), which only shows data of a subset of the students, while some participants thought that it was performing some kind of advanced analysis.

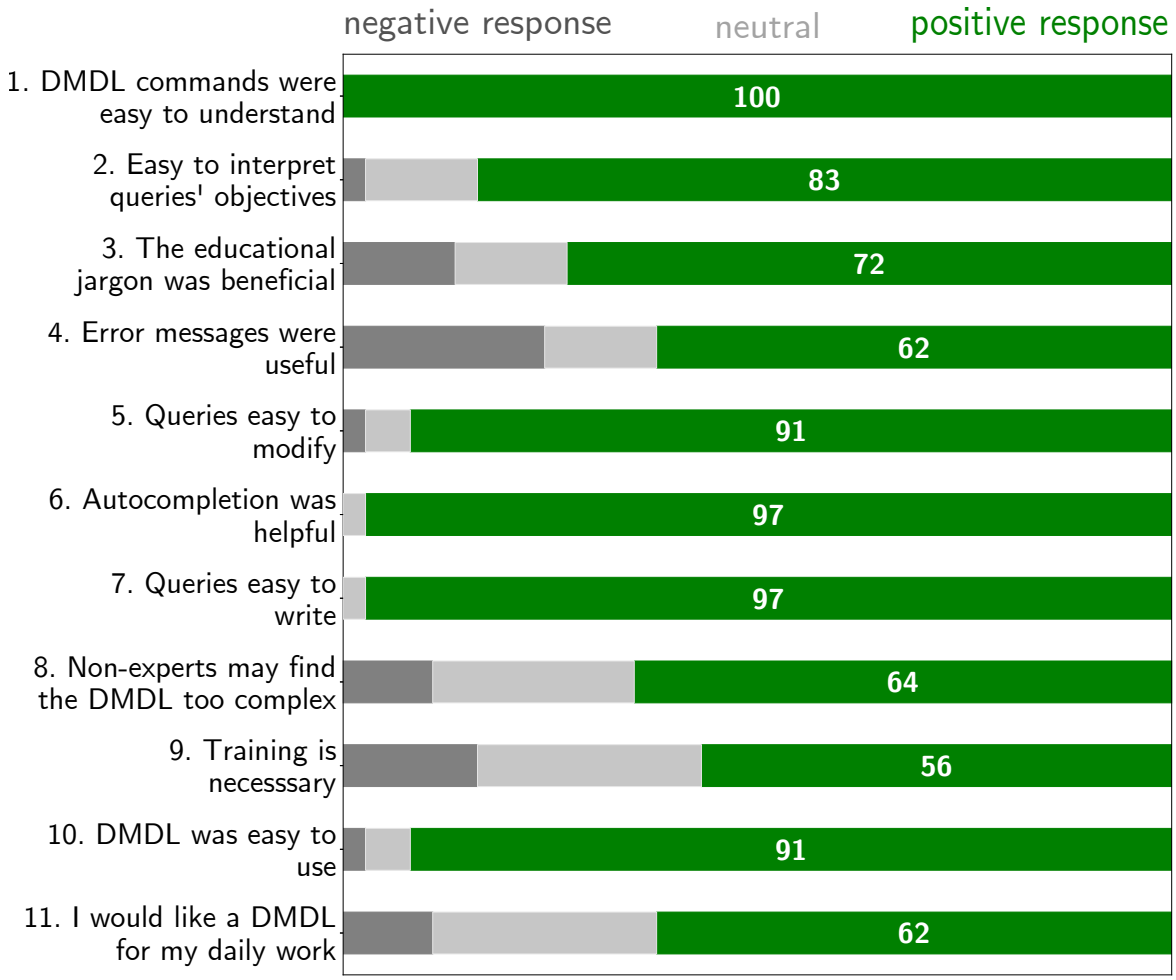


Fig. 6.4 Likert-scale responses of the post-questionnaire.

- Using the wrong entity to invoke an analysis. In some of the questions, participants had to only analyse either the demographic or the activity data of the students, but they selected the wrong entity when formulating the query.
- Similar to the previous one, and as commented in the above highlights, using the wrong attribute when formulating an analysis query or when gathering a value from the results was also a detected as a mistake.

Next section comments on the subjective opinion of the participants in the usefulness of the DMDL prototype.

6.5.3 Post: Participants' Opinion

The results of the satisfaction questionnaire filled by the participants after the test are shown in Figure 6.4. Positive responses, these are, those ones that are positive from the point of view of the DMDL, appear on the right side, green-filled, and with their percentages shown in white color. Neutral responses are indicated in light grey, and negative responses appear on the left in a darker grey.

Almost all participants considered that the objective of commands and written queries were easy to understand (statements 1 and 2). Additionally, most participants manifested that the use of a vocabulary close to their domain was beneficial for this understanding (statement 3). On the other hand, a few participants disagreed with this last statement, claiming that the domain was irrelevant (1 participant), that the main reason was the use of natural language-like syntax (2 participants), or that the importance was in understanding the commands' objective (1 participant). Two of the participants also mentioned that they would need to try a DMDL focused in an unfamiliar topic to be able to properly answer this query.

The opinion on the error messages was more divided (statement 4). A few participants recognised that they did not pay attention to the error messages, and that in those cases where an error was shown they tried to rewrite the query completely, as queries were small and the autocompletion system made this rewriting faster. Other participants fixed their errors by using the provided printed documentation instead of these error messages. For more than 20% of the participants, the experience with this feature was negative. The reason for this resides in the maturity of the provided DMDL prototype: although a good amount of effort was invested in providing as many friendly error messages as possible, some of the error messages that remained were the ones generated by default by Xtext, i.e., the tool we used to implement this editor. These messages are domain-agnostic technical errors related to syntax structural problems found by the query parser. Therefore, these messages offered bad assistance to participants non-familiar with computer languages during the process of writing a query. Nevertheless, more than 60% of users considered the validator a good feature. As future work, we will try to provide more user-friendly and informative error messages.

When modifying or refining an existent query (statement 5), those participants who struggled had most of their problems when the change was in the middle of such a query (e.g. in a command parameter, or in the employed entity name). The rewriting of the query described in the previous paragraph was also applied in this case. Again, these

issues were only experienced by a minority of the participants, as the vast majority of them (∼90%) considered that refinements were easy to include.

The autocompletion system was considered a very useful feature by all participants (statement 6). This system, along with the received training and the printed help material, made writing queries from scratch a feasible task (statement 7).

More than 60% of users considered that non-expert users could learn to use this kind of languages (statement 8). Additionally, among those that indicated the opposite in the Likert scale, we detected that some participants had marked the opposite value with respect to the provided rationale about the selected value (remember that statement 8 was the only one where a disagreement value is a positive answer for the test, so that could have confused these participants.). We did not alter these responses, but it probably means that the number of participants considering that non-experts could learn to use a DMDL is slightly larger than 60%.

Around half of the participants considered necessary to receive some training to properly use the DMDL (∼55%, statement 9). Those who did not commented that, for instance, the given documentation was enough to understand how to use the language.

As a general agreement, 90% of the participants indicated that the provided DMDL was easy to use (statement 10). Those who disagreed mentioned that the problems they encountered when using the DMDL would get mitigated with a longer training period.

Lastly, 60% of the particpants considered that a DMDL adapted to their domain could be an useful tool to assist them with their activities (statement 11). Some of the comments provided by those who did not see useful the existence of such a DMDL related to the absence of data analysis in their concrete field, e.g., Mathematics teachers whose research is based in performing demonstrations or in solving differential equation systems did not find DMDLs useful for their field, as it is natural.

To conclude, users could give some general opinions about the language and the experiments in a free-text field. We found the following two issues the most interesting ones:

- Some participants were curious about how data could be introduced into the system, e.g., to use their own courses data to launch questions. This is directly linked with the use of Lavoisier, so we hope to include this language into the user tests in future experiments.

- Also, a few participants manifested their interest in knowing the background analysis processes that were being executed to try to understand better what was

happening. For instance, some participants mentioned that they would like to know the concrete correlation technique employed when calculating the relation between two columns of the dataset, e.g., *Pearson*, *Kendall* or *Spearman*. It could be interesting to investigate the development of *white-box* analysis processes, i.e., processes that try to explain to the non-expert *how* they are analysing the data in some sort of high-level language.

In conclusion, participants' subjective opinions confirm our previous findings, and we can state that experts of the educational domain can use our DMDL to execute data mining tasks. Moreover, participants consider that experts of other domains should be able to use similar DMDLs although, from a scientific point of view, we cannot claim this without further experiments.

6.5.4 Results Summary

The pre-questionnaire information shows that the subjects that participated in these experiments have a heterogeneous technical background. Although most of them use office and spreadsheet apps, only a handful use advanced analysis tools, either through professional software or by directly employing specialised languages and libraries.

Most of the participants completed correctly the data mining tasks we proposed, which indicates that the DMDL was understood, and that they could employ it. The test was harder for those participants with a less technological background, i.e., Education teachers. Nevertheless, the results were still very good in their case.

The opinion given by the participants in the post-questionnaire matches the results of the tests. These participants found the provided DMDL easy to use and understand, and a good portion of them considered the existence of such a language to assist them in their application domain an interesting option.

6.6 Threats to Validity

This section discusses potential threats to the validity of the conclusions we reached after performing our experiments. Specifically, we identified the following threats:

1. The use of synthetic data.
2. The effect of the training session.
3. False positives on the tests.

4. Generalisation of results.
5. Nature of the case study.
6. Nature of the proposed analysis tasks.
7. Number of case studies.

We comment on these threats in the following.

Use of synthetic data. As mentioned in Section 6.4.3, despite having real data, and due to personal data protection laws, we had to use synthetic data in our experiments. Nevertheless, it should be noted that we were not analysing whether our DMDLs are able of finding useful facts in real data, but if domain experts are able to use a DMDL over real data. To this respect, our synthetic data mimics the structure of data used in the educational data mining domain (Jugo et al. [84], Romero and Ventura [140], Zorrilla and García-Saiz [176]). Therefore, our domain experts had to face the same problems as if they were working with the real data.

Effect of the training session. As a second threat to validity, it might be that, during the training session, we solved very similar problems to the ones that domain experts, i.e., teachers, had to solve during the test. Therefore, teachers could have solved some tasks simply by imitation, replicating steps they remembered from the training sessions. This would imply that teachers were able to use our DMDL, although they did not understand how it actually works. To alleviate this threat, we changed the domain of the training session to a bank marketing analysis example, so that the details of the educational domain were received for the first time in the test session. Additionally, we ensured that some time, at least two hours, passed between the training and the test sessions, to allow participants to disconnect their minds between both sessions. Also, it is important to remark that participants did not have any incentive in scoring a high grade in the test, as this test was anonymous, and they would not get any benefit from cheating the questions.

False positives in the tests. The results of some analysis tasks were just a number, or a choice among a reduced number of options. Therefore, a teacher could have provided a correct answer by accident, instead of as a result of a correct process. To detect and remove these false positives, we not only collected the simple answer of each task, but also some extra information, such as the formulated queries. This helped us to detect these false positives. As an example, in one of the queries some participants used the wrong data for the analysis (e.g. demographic instead of activity data), but the result returned by the analysis was coincidentally the same as with

the correct entity. As we had the query they used to answer the question, we could correctly identify the mistake and mark these answers as wrong.

Generalisation of results. Certain characteristics of the participants could have made the results of these experiments not generalisable to a wider population. For instance, if we had used just Biology professors, we could not claim that our results also apply to teaching assistants in History. To avoid this, we selected participants of different kind (e.g. teachers and graduate students) and belonging to different knowledge areas. Also, these participants had heterogeneous technical backgrounds, e.g., previous experience with programming languages, or even in computer skills. Therefore, we believe that the experiment participants offer a good spectrum of different technical knowledge levels and, as a result of that, the conclusions of this experiment should at least be generalisable to the university educational level. Moreover, we did not detect any specific advantage of the university context in using the DMDL that might make the conclusions of these experiments not extensible to other educational levels, such as high schools.

Nature of the case study or the proposed tasks. It might be argued that our findings are due to some characteristics of the used data or the proposed questions, but that other data or different questions might have led us to different conclusions. In particular, it can be claimed that the set of proposed tasks is quite short. This statement is absolutely right. First of all, the experiment was intentionally designed to be short, in order to encourage participation. It must be highlighted that participants volunteered to participate in these experiments, and that they were not rewarded in any way. So, we did not want to take too much time from them. Secondly, we have not detected any prominent issue in the case study that might be biasing our results. Nevertheless, we plan to replicate these experiments with the same participants but with different data and analysis tasks as part of our future work.

Only one case study. Another external threat about generalisation comes from the fact that our experiments only included one case study: a DMDL for the educational domain used to answer one set of questions. The absence of more case studies was provoked by our limited time frame to prepare these experiments. As commented, we tried to mitigate this issue by using this case study in several experiments with heterogeneous participants, but the inclusion of more case studies would have been preferred. We will expand these experiments in the future with new sets of questions in the same educational domain, or with new DMDLs applied to new contexts.

6.7 Chapter Summary

This chapter has shown our efforts to evaluate whether DMDLs created with the FLANDM framework meet our initial goals.

We started by comparing the features offered by our DMDLs with existing approaches from the state of the art. This comparison shows that DMDLs match the benefits offered by these approaches, and in some cases they provide better results for some quality attributes, such as granting users the ability to fine-tune the executed data mining processes without requiring the intervention of a data scientist.

Our evaluation actions continued by analysing how well DMDLs support a set of general usability heuristics defined by Molich and Nielsen [117]. In this analysis, it was concluded that the different assistance features provided by our DMDLs offered a good support for these heuristics.

As the next step, we performed empirical experiments with one of our DMDLs. The DMDL prototype used in the experiments is an updated version of the educational DSL that we devised as our first contribution to the data mining democratisation field. In these experiments, university teachers and graduate students had the opportunity to test our DMDL prototype by performing some analysis tasks over data from a university course. The results of these experiments show that most participants were able to correctly complete these tasks with the assistance of the DMDL. In addition, these participants later manifested that the provided DMDL was understandable, and that the analysis queries were easy to write and interpret.

The effort invested to carry out these experiments only allowed us to test a DMDL from the FLANDM framework. In the future, we plan to extend these empirical experiments for more DMDLs set up for other contexts, and to include end-user validation of Lavoisier and Pinset.

Chapter 7

Summary and Future Work

7.1 Thesis Summary

Data mining techniques allow finding useful insights, previously undetected, in data from a domain. These techniques, however, cannot be used by average decision makers, these are, people who are experts in a domain but who, in most cases, do not have the knowledge that is required to properly apply these analysis techniques. This knowledge involves different advanced aspects, such as statistics, data manipulation, or the definition of algorithms with one or more programming languages.

Data Mining Democratisation is a field that aims to make data mining techniques usable by people without deep knowledge in these technical aspects. In this thesis, we studied the status of this field, and tried to overcome some of its current issues through the application of Model-Driven Engineering and Domain-Specific Languages technologies. These technologies seemed ideal for this purpose, as they allow increasing the abstraction level of an application, so that low-level details can be omitted from the end user's perspective if desired. In addition, these abstractions can contribute to reducing development costs, as implementation-dependent details are only a concern at the last step of the development process.

As almost any research methodology recommends, we started our work by analysing the state of the art of the data mining democratisation field. This analysis was performed by means of a systematic review, which encompassed both works from the academia and available applications from the state of the art. This combination allowed us to get a holistic view of the field, and to identify some of its current shortcomings. In this thesis, we focused on two of these shortcomings: (1) the importance and cost of developing easy-to-use analysis solutions customised for each application domain; and

(2) the scarcity of existing solutions to help with the data selection and transformation tasks.

We targeted the first shortcoming by developing high-level DSLs for the execution of data mining processes, which we denoted as *Data Mining Democratisation Languages (DMDLs)*. These DMDLs shield end users from any technical details of the analysis processes executed in the background, so that users do not need to have deep knowledge in the techniques applied in such processes. Both the syntax and analysis processes of each DMDL are adapted to its concrete domain, e.g., educational, medical, software development, among others. Therefore, these DMDLs are devised to feel familiar to end users, and also their analyses are prepared to take into account details of the domain, so the accuracy of these analyses is not compromised. To alleviate the cost of developing a different DMDL for each concrete domain, we designed a framework, *FLANDM (Framework to develop LANguages for Data Mining)*, which allows for an easy instantiation of DMDLs with reduced costs.

The described DMDLs operate over tabular datasets, which need to be created by data scientists from the available data in the domain. For this creation task, the input of a domain expert can be very valuable, as these experts have a lot of knowledge of the meaning, implications, and intrinsic relationships present in data that may not be detected by a data scientist. As the second shortcoming we detected in our review, not that many works exist to try help end users participate in this task. To help with this issue, we devised *Lavoisier*, a DSL that provides end users with an adapted syntax to select the bundle of data be included in an analysis from a high-level conceptual model of the available domain data. Users of this language only have to worry about selecting the data that they consider interesting for an analysis, and Lavoisier automatically transforms this selection into a tabular dataset that can be digested by conventional analysis tools and libraries.

Continuing with the second shortcoming, we detected that Lavoisier's syntax was not adequate to perform certain advanced calculations, such as aggregates over the provided conceptual model. These computations required a syntax closer to the constructs that are offered by conventional programming languages such as C or Java. To avoid increasing the complexity of Lavoisier by including these constructs in its syntax, we developed a complementary language, called *Pinset*, that offers these constructs, and as such it is more oriented to users with computer programming skills.

We evaluated our contributions throughout their development, e.g., the reduction in development or maintenance costs offered by FLANDM (see Section 3.5), or the better conciseness of Lavoisier and Pinset when compared with existing tools for the

same objectives (see Sections 4.6 and 5.6.4, respectively). Additionally, we started the end-user evaluation of our DSLs through some empirical experiments, as described in Section 6.4.

We believe that our MDE and DSL-based contributions have improved the current state of the Data Mining Democratisation field. Next section presents a summary of these contributions, and Section 7.3 describes possible future works to continue with this research.

7.2 Thesis Contributions

Most contributions of this thesis are already published in peer-reviewed conferences and journals (see List of Publications on page xxv). The following highlights summarize the outcomes of these contributions:

- A systematic review involving around 700 tools and academic works in data mining democratisation.
- A detailed analysis of 28 tools and 15 works identified in this review.
- A grouping of these works in four categories, according to their strategies.
- A study of these categories, to identify their strengths and weaknesses.
- The identification of a set of current shortcomings in the data mining democratisation field, to serve as guide for future research.
- The development of *Domain-Specific Languages for Data Mining Democratisation (DMDLs)*, for non-expert users to apply data mining techniques to data from their domains.
- The definition of a framework, *FLANDM*, that reduces the development of a DMDL by 50%.
- The creation of several DMDLs for different domains, to validate the benefits claimed by FLANDM.
- The formal definition of a flattening operator, which allows selecting information from an object-oriented domain model and automatically transforms this information into a dataset ready for analysis.

- A high-level DSL, Lavoisier, that internally employs this operator to allow decision makers to actively participate in the data selection and data preparation tasks of a data mining process.
- A conciseness and complexity demonstration against state-of-the-art tools to perform this data preparation tasks, where Lavoisier provided better results than these tools.
- A DSL for power users, Pinset, which combines primitives specific to the dataset extraction task with conventional software programming constructs such as conditions or loops, for the advanced extraction of datasets from models.
- A comparison of Pinset against general-purpose model transformation languages, which assessed the benefits of having a DSL for the dataset extraction task.
- A comparison of the developed DMDLs against solutions for the state of the art, to assess whether some of the shortcomings detected in the review were mitigated.
- A set of empirical tests with an educational DMDL, where university teachers were able to launch data mining processes to analyse courses data.

Finally, as the last contribution of this thesis, in the next section we present some possible research lines to continue this work in the future.

7.3 Future Work

There are several ways to continue the lines of work started during this thesis, and some new lines that we would like to study in detail. These can be summarised as follows:

- We will increase the empirical experiments with DMDLs, to make them one of the fundamental indicators of how to improve the FLANDM framework, as some authors defend (Barisic et al. [13]). More tests will be included in the same educational domain, to answer different questions and to analyse other bundles of data. Also, DMDLs for new domains will be developed and tested.
- Lavoisier and Pinset will be included in the end-user experiments. These experiments will help assess whether Lavoisier constructs can be easily understood

and applied by non-expert users, and determine if Pinset presents advantages to power users (e.g. software programmers) over the tools that these users employ daily. Any shortcomings detected in these experiments would help with the improvement of these languages.

- Our initial experiments with a DMDL showed that non-experts are able to use them. Apart from selecting the data bundle and the analysis to perform, this DMDL did not include any technical parameters to customise the data mining processes executed in the background. We will study how to give end users a more fine-grained control of these processes by the inclusion of some configurable parameters, while at the same time maintaining the DMDLs benefit of being easy to use.

- While performing the experiments, some participants were curious about the underlying data mining processes being executed. To make this execution more transparent, we will work in different ways to explain non-expert users the analysis processes being carried out, so that we advance from black-box processes to a white- or open-box processes approach.

- It is difficult to provide encapsulated data mining processes that can work properly with different datasets and contexts, which provokes that changes in these processes are required when moving to a new domain. To reduce the need of data scientists when creating a new DMDL, we will study how some promising techniques, such as meta-learning (Lemke et al. [103]) or parameter-less algorithms (Zorrilla et al. [177]), can help define more autonomous and auto-configurable analysis processes.

- One of the detected shortcomings of Lavoisier was the lack of support for calculating complex derived values. We will try to include this support without increasing the complexity of the other constructs of the language.

- As for the Pinset language, new features will also be worked on, such as improved support for extracting datasets from batches of models, instead of a single one. Also, as this language is more oriented towards advanced users such as software programmers, we believe that performance is a relevant indicator to convince these users to employ this language. Therefore, we will include a performance study in our future work with this language.

- We will also investigate how to extend the support of Lavoisier and Pinset for other conceptual models apart from object-oriented models, such as ontologies or multidimensional models.
- Lastly, we will put the contributions of this thesis to test in new and emergent domains where data mining might play a key role, such as Ambient Intelligence, Smart Cities or Industry 4.0.

References

[1] A Abelló, J Darmont, L Etcheverry, M Golfarelli, J N Mazón, F Naumann, T B Pedersen, S Rizzi, J Trujillo, P Vassiliadis, and G Vossen. Fusion cubes: Towards self-service business intelligence. *International Journal of Data Warehousing and Mining*, 9:66–88, 2013. ISSN 15483924. doi:10.4018/jdwm.2013040104.

[2] Bram Adams, Ryan Kavanagh, Ahmed E. Hassan, and Daniel M. German. An empirical study of integration activities in distributions of open source software. *Empirical Software Engineering*, 21(3):960–1001, Jun 2016. ISSN 1573-7616. doi:10.1007/s10664-015-9371-y.

[3] Samuel A. Ajila and Di Wu. Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, 80(9):1517 – 1529, 2007. ISSN 0164-1212. doi:10.1016/j.jss.2007.01.011.

[4] J.M. Alonso and C. Mencar. Building cognitive cities with explainable artificial intelligent systems. *CEUR Workshop Proceedings*, 2071, 2018. ISSN 16130073.

[5] Paul Alpar and Michael Schulz. Self-Service Business Intelligence. *Business & Information Systems Engineering*, 58(2):151–155, apr 2016. ISSN 2363-7005. doi:10.1007/s12599-016-0424-6.

[6] D. Ameller and et al. Dealing with non-functional requirements in model-driven development: A survey. *IEEE Transactions on Software Engineering*, pages 1–1, 2019. ISSN 0098-5589. doi:10.1109/TSE.2019.2904476.

[7] Mihael Ankerst, Martin Ester, and Hans-Peter Kriegel. Towards an effective cooperation of the user and the computer for classification. *Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining (KDD)*, pages 179–188, 2000. doi:10.1145/347090.347124.

[8] Paolo Atzeni, Paolo Cappellari, Riccardo Torlone, Philip A. Bernstein, and Giorgio Gianforme. Model-independent schema translation. *VLDB Journal*, 17 (6):1347–1370, 2008. ISSN 10668888. doi:10.1007/s00778-008-0105-2.

[9] Önder Babur, Loek Cleophas, and Mark van den Brand. Hierarchical Clustering of Metamodels for Comparative Analysis and Visualization. In *Modelling Foundations and Applications*, volume 7949, pages 3–18. 2016. ISBN 978-3-642-39012-8. doi:10.1007/978-3-319-42061-5_1.

[10] John W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich ACM-GAMM conference. In *IFIP Congress*, pages 125–131, 1959.

[11] Paul Baker, Shiou Loh, and Frank Weil. Model-Driven Engineering in a Large Industrial Context — Motorola Case Study. In Lionel Briand and Clay Williams, editors, *Model Driven Engineering Languages and Systems*, pages 476–491, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-32057-9. doi:10.1007/11557432_36.

[12] Imad Bani-Hani, Olgerta Tona, and Sven Carlsson. From an information consumer to an information author: a new approach to business intelligence. *Journal of Organizational Computing and Electronic Commerce*, 28(2):157–171, apr 2018. ISSN 1091-9392. doi:10.1080/10919392.2018.1444358.

[13] Ankica Barisic, Vasco Amaral, and Miguel Goulão. Usability driven DSL development with USE-ME. *Computer Languages, Systems & Structures*, 51:118–157, 2018. doi:10.1016/j.cl.2017.06.005.

[14] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Automated clustering of metamodel repositories. In *Advanced Information Systems Engineering - 28th International Conference, CAiSE 2016*, pages 342–358, 2016. doi:10.1007/978-3-319-39696-5_21.

[15] V. R. Basili and H. D. Rombach. The tame project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14 (6):758–773, June 1988. ISSN 0098-5589. doi:10.1109/32.6156.

[16] Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0321146530.

[17] Michael Behringer, Pascal Hirmer, and Bernhard Mitschang. Towards interactive data processing and analytics putting the human in the center of the loop. *Proceedings of the 19th International Conference on Enterprise Information Systems - ICEIS 2017*, 3(Iceis):87–96, 2017. doi:10.5220/0006326300870096.

[18] Lynn Beighley. *Head first SQL*. O'Reilly, 2007. ISBN 978-0-596-52684-9.

[19] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 447–450, May 2017. doi:10.1109/MSR.2017.24.

[20] M Ben Ayed, H Ltifi, C Kolski, and A M Alimi. A user-centered approach for the design and implementation of KDD-based DSS: A case study in the healthcare domain. *Decision Support Systems*, 50:64–78, 2010. ISSN 01679236. doi:10.1016/j.dss.2010.07.003.

[21] Besim Bilalli, Alberto Abelló, Tomàs Aluja-Banet, and Robert Wrembel. Intelligent assistance for data pre-processing. *Computer Standards & Interfaces*, 57: 101–109, 2018. doi:10.1016/j.csi.2017.05.004.

[22] B. W. Boehm. Improving software productivity. *Computer*, 20(9):43–57, Sept 1987. ISSN 0018-9162. doi:10.1109/MC.1987.1663694.

[23] Marc Boullé, Clément Charnay, and Nicolas Lachiche. A scalable robust and automatic propositionalization approach for Bayesian classification of large mixed numerical and categorical data. *Machine Learning*, 2018. ISSN 0885-6125. doi:10.1007/s10994-018-5746-9.

[24] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition, 2012. ISBN 1608458822, 9781608458820. doi:10.2200/S00441ED1V01Y201208SWE001.

[25] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008. ISBN 3540732624, 9783540732624.

[26] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4): 571 – 583, 2007. ISSN 0164-1212. doi:10.1016/j.jss.2006.07.009.

[27] C. Couto et al. Predicting software defects with causality tests. *Journal of Systems and Software*, 93:24 – 41, 2014. ISSN 0164-1212. doi:10.1016/j.jss.2014.01.033.

[28] M M Campos, P J Stengard, and B L Milenova. Data-centric automated data mining. In *ICMLA 2005: 4th International Conference on Machine Learning and Applications*, pages 97–104, 2005. ISBN 0769524958 (ISBN); 9780769524955 (ISBN). doi:10.1109/ICMLA.2005.18.

[29] Longbing Cao. Domain-Driven Data Mining: Challenges and Prospects. *IEEE Transactions on Knowledge and Data Engineering*, 22(6):755–769, jun 2010. doi:10.1109/TKDE.2010.32.

[30] Longbing Cao. Data science and analytics: a new era. *International Journal of Data Science and Analytics*, 1(1):1–2, apr 2016. doi:10.1007/s41060-016-0006-1.

[31] Jesús Caro-Gutiérrez, Oscar M. Peréz-Landeros, Félix F. González-Navarro, Mario A. Curiel-Álvarez, Benjamín Valdez-Salas, and Nicola Radnev-Nedev. Data mining to predict the average outer diameter of vertically aligned tio2 nanotubes. *Computational Materials Science*, 162:82 – 87, 2019. ISSN 0927-0256. doi:10.1016/j.commatsci.2019.02.041.

[32] Jian Chen, Manar H. Alalfi, Thomas R. Dean, and S. Ramesh. Modeling autosar implementations in simulink. In *Modelling Foundations and Applications*, pages 279–292. Springer International Publishing, 2018. ISBN 978-3-319-92997-2. doi:10.1007/978-3-319-92997-2_18.

[33] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Soft. Eng.*, 20:476–493, 1994. ISSN 0098-5589. doi:10.1109/32.295895.

[34] Luca Chittaro, Carlo Combi, and Giampaolo Trapasso. Data mining on temporal data: a visual approach and its clinical application to hemodialysis. *Journal of Visual Languages & Computing*, 14(6):591–620, 2003. ISSN 1045926X. doi:10.1016/j.jvlc.2003.06.003.

[35] David Raymond Christiansen, Klaus Grue, Henning Niss, Peter Sestoft, and Kristján S. Sigtryggsson. An actuarial programming language for life insurance and pensions. 2013.

[36] B. Combemale, J. DeAntoni, B. Baudry, R. B. France, J. Jezequel, and J. Gray. Globalizing modeling languages. *Computer*, 47(06):68–71, jun 2014. ISSN 0018-9162. doi:10.1109/MC.2014.147.

[37] Benoit Combemale, Robert France, Jean-Marc Jézéquel, Bernhard Rumpe, Jim R.H. Steel, and Didier Vojtisek. *Engineering Modeling Languages*. Chapman and Hall/CRC, 2016. ISBN 9781466583733.

[38] Sven F. Crone, Stefan Lessmann, and Robert Stahlbock. The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, 173(3):781 – 800, 2006. ISSN 0377-2217. doi:10.1016/j.ejor.2005.07.023.

[39] C Cunningham. PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS. *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 998–1009, 2004.

[40] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 31–41, May 2010. doi:10.1109/MSR.2010.5463279.

[41] Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez. Towards a DSL for Educational Data Mining. In *Languages, Applications and Technologies*, pages 79–90, 2015. doi:10.1007/978-3-319-27653-3_8.

[42] Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez. On the Automated Transformation of Domain Models into Tabular Datasets. *ER FORUM*, 1979, 2017. ISSN 16130073.

[43] Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez. FLANDM: a development framework of domain-specific languages for data mining democratisation. *Computer Languages, Systems and Structures*, 54: 316–336, 2018. ISSN 14778424. doi:10.1016/j.cl.2018.07.002.

[44] Alfonso de la Vega, Pablo Sanchez, and Dimitris Kolovos. Pinset: A DSL for Extracting Datasets from Models for Data Mining-Based Quality Analysis. *Quality of Information and Communications Technology (QUATIC)*, pages 83–91, 2018. doi:10.1109/quatic.2018.00021.

[45] Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez. How Far are we from Data Mining Democratisation? A Systematic Review. *arXiv e-prints*, art. arXiv:1903.08431, Mar 2019. URL https://arxiv.org/abs/1903.08431.

[46] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. doi:10.1111/j.2517-6161.1977.tb01600.x.

[47] Arie Van Deursen and Paul Klint. Little languages: little maintenance? *Journal of Software Maintenance: Research and Practice*, 10(2):75–92, 1998. ISSN 1040-550X. doi:10.1002/(SICI)1096-908X(199803/04)10:2<75::AID-SMR168>3.0.CO;2-5.

[48] Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Mining Metrics for Understanding Metamodel Characteristics. In *International Workshop on Modeling in Software Engineering, MiSE 2014*, pages 55–60, 2014. ISBN 978-1-4503-2849-4. doi:10.1145/2593770.2593774.

[49] Laurent Doldi. *Validation of Telecom Systems with SDL: The Art of SDL Simulation and Reachability Analysis*. Wiley, April 2003.

[50] Gaia Donati and Chris Woolston. Information Management: Data Domination. *Nature*, 548:613–614, 2017. doi:10.1038/nj7669-613a.

[51] Nicole Dungca. In first, Uber to share ride data with Boston. The Boston Globe; http://bit.ly/1NrvUhm, January 2015. [Online; accessed March 2019].

[52] Roberto Espinosa, Diego García-Saiz, Marta Zorrilla, Jose Jacobo Zubcoff, and Jose-Norberto Mazón. Enabling non-expert users to apply data mining for bridging the big data divide. In *Data-Driven Process Discovery and Analysis, International Symposium, SIMPDA 2013*, pages 65–86, 2015. ISBN 978-3-662-46436-6. doi:10.1007/978-3-662-46436-6_4.

[53] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 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*, KDD'96, pages 226–231. AAAI Press, 1996.

[54] Eric Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[55] Moritz Eysholdt and Heiko Behrens. Xtext: Implement Your Language Faster Than the Quick and Dirty Way. In *Companion to the 25th Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (SPLASH/OOPSLA)*, pages 307–309, 2010. ISBN 978-1-4503-0240-1. doi:10.1145/1869542.1869625.

[56] F. Palomba et al. Investigating Code Smell Co-Occurrences Using Association Rule Learning: A Replicated Study. In *IEEE Workshop on ML Techniques for Soft. Quality Evaluation (MaLTeSQuE)*, pages 8–13, 2017. doi:10.1109/MALTESQUE.2017.7882010.

[57] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering*, 23(1):452–489, 2018. doi:10.1007/s10664-017-9523-3.

[58] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2755–2763, Cambridge, MA, USA, 2015. MIT Press.

[59] Iztok Jr. Fister, Tomaž Kosar, Iztok Fister, and Marjan Mernik. Easytime++: A Case Study Of Incremental Domain-Specific Language Development. *Information Technology And Control*, 42(1), mar 2013. ISSN 1392-124X. doi:10.5755/j01.itc.42.1.1968.

[60] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0321127420.

[61] Martin Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010. ISBN 0321712943, 9780321712943.

[62] JE Gaffney and TA Durek. Software reuse — key to enhanced productivity: some quantitative models. *Information and Software Technology*, 31(5):258–267, jun 1989. ISSN 09505849. doi:10.1016/0950-5849(89)90005-0.

[63] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, October 1994.

[64] E García, C Romero, S Ventura, and C De Castro. A collaborative educational association rule mining tool. *Internet and Higher Education*, 14:77–88, 2011. ISSN 10967516. doi:10.1016/j.iheduc.2010.07.006.

[65] Jill Gerhardt-Powals. Cognitive engineering principles for enhancing human-computer performance. *International Journal of Human-Computer Interaction*, 8(2):189–211, 1996. doi:10.1080/10447319609526147.

[66] Matteo Golfarelli and Stefano Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill Education, 1 edition, 6 2009. ISBN 9780071610391.

[67] Thomas Guyet, Catherine Garbay, and Michel Dojat. Knowledge construction from time series data using a collaborative exploration system. *Journal of Biomedical Informatics*, 40(6):672–87, 2007. ISSN 1532-0480. doi:10.1016/j.jbi.2007.09.006.

[68] H. Osman et al. Automatic feature selection by regularization to improve bug prediction accuracy. In *IEEE Workshop on ML Techniques for Soft. Quality Evaluation (MaLTeSQuE)*, pages 27–32, 2017. doi:10.1109/MALTESQUE.2017.7882013.

[69] Jean-Luc Hainaut. The Transformational Approach to Database Engineering. In *Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal*, pages 95–143. Springer, 2006. doi:10.1007/11877028_4.

[70] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. . The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. doi:10.1145/1656274.1656278.

[71] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011. ISBN 978-0123814791. URL http://hanj.cs.illinois.edu/bk3/.

[72] Zhao Han and Carson K Leung. FIMaaS. In *Proceedings of the 2015 International Conference on Big Data Applications and Services - BigDAS '15*, pages 84–91. ACM Press, 2015. ISBN 9781450338462. doi:10.1145/2837060.2837072.

[73] Daniel N. Hill, Houssam Nassif, Yi Liu, Anand Iyer, and S. V. N. Vishwanathan. An efficient bandit algorithm for realtime multivariate optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1813–1821, 2017. doi:10.1145/3097983.3098184.

[74] Georg Hinkel, Thomas Goldschmidt, Erik Burger, and Ralf Reussner. Using internal domain-specific languages to inherit tool support and modularity for model transformations. *Software & Systems Modeling*, Jan 2017. ISSN 1619-1374. doi:10.1007/s10270-017-0578-9.

[75] Richard Holcomb and Alan L. Tharp. What users say about software usability. *International Journal of Human–Computer Interaction*, 3(1):49–78, 1991. doi:10.1080/10447319109525996.

[76] George Hripcsak. Writing arden syntax medical logic modules. *Computers in Biology and Medicine*, 24(5):331 – 363, 1994. ISSN 0010-4825. doi:10.1016/0010-4825(94)90002-7.

[77] John Hutchinson, Jon Whittle, and Mark Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, 89:144 – 161, 2014. ISSN 0167-6423. doi:10.1016/j.scico.2013.03.017.

[78] Claudia Imhoff and Colin White. Self-Service Business Intelligence: Empowering Users to Generate Insights. *TDWI Best Practices Report*, 2011.

[79] Peter Zilahy Ingerman. Panini-Backus Form; Suggested. *Commun. ACM*, 10(3): 137–, March 1967. ISSN 0001-0782. doi:10.1145/363162.363165.

[80] S Jalali and C Wohlin. 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)*, pages 29–38, 2012. ISBN 9781450310567. doi:10.1145/2372251.2372257.

[81] Monica Johar, Vijay Mookerjee, and Suresh Sethi. Optimal software design reuse policies: A control theoretic approach. *Information Systems Frontiers*, 17(2): 439–453, Apr 2015. ISSN 1572-9419. doi:10.1007/s10796-013-9421-1.

[82] Magne Jorgensen and Martin Shepperd. A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, jan 2007. ISSN 0098-5589. doi:10.1109/TSE.2007.256943.

[83] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A Model Transformation Tool. *Sci. Comput. Program.*, 72:31–39, 2008. ISSN 0167-6423. doi:10.1016/j.scico.2007.08.002.

[84] Igor Jugo, Božidar Kovačić, and Edvard Tijan. Cluster analysis of student activity in a web-based intelligent tutoring system. *Pomorstvo: Scientific Journal of Maritime Research*, 29(1):75–83, 2015.

[85] M R Kamdar, D Zeginis, A Hasnain, S Decker, and H F Deus. ReVeaLD: A user-driven domain-specific interactive search platform for biomedical research. *Journal of Biomedical Informatics*, 47:112–130, 2014. ISSN 15320464. doi:10.1016/j.jbi.2013.10.001.

[86] B Kamsu-Foguem, G Tchuente-Foguem, L Allart, Y Zennir, C Vilhelm, H Mehdaoui, D Zitouni, H Hubert, M Lemdani, and P Ravaux. User-centered visual analysis using a hybrid reasoning architecture for intensive care units. *Decision Support Systems*, 54:496–509, 2012. ISSN 0167-9236. doi:10.1016/j.dss.2012.06.009.

[87] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990. URL http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231.

[88] J. M. Kanter and K. Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, Oct 2015. doi:10.1109/DSAA.2015.7344858.

[89] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[90] Jinhan Kim, Sanghoon Lee, Seung-Won Hwang, and Sunghun Kim. Enriching documents with examples: A corpus mining approach. *ACM Trans. Inf. Syst.*, 31(1):1:1–1:27, January 2013. ISSN 1046-8188. doi:10.1145/2414782.2414783.

[91] Barbara Kitchenham and Stuart Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering (Version 2.3). EBSE 2007-001, 2007.

[92] S Klenk, J Dippon, P Fritz, and G Heidemann. Interactive survival analysis with the OCDM system: From development to application. *Information Systems Frontiers*, 11:391–403, 2009. ISSN 13873326. doi:10.1007/s10796-009-9152-5.

[93] Anneke Kleppe. A Language Description is More than a Metamodel. In *Models in Software Engineering*, number 612, pages 28–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-69073-3_4.

[94] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008. ISBN 0321553454, 9780321553454.

[95] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional, April 2003.

[96] Arno Jan Knobbe, Marc De Haas, and Arno Siebes. Propositionalisation and Aggregates. *Principles of Data Mining and Knowledge Discovery*, 2168:277–288, 2001. ISSN 16113349. doi:10.1007/3-540-44794-6_23.

[97] Ron Kohavi, Dan Sommerfield, and James Dougherty. Data Mining using MLC++, A Machine Learning Library in C++. *International Journal on Artificial Intelligence Tools*, 6(4):537–566, nov 1996. doi:10.1142/S021821309700027X.

[98] Dimitrios S Kolovos, Richard F Paige, and Fiona A C Polack. The Epsilon Object Language (EOL). In *Model Driven Architecture – Foundations and Applications*, pages 128–142, 2006. ISBN 978-3-540-35910-4. doi:10.1007/11787044_11.

[99] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. Polack. The Epsilon Transformation Language. In *Proceedings of the 1st International Conference on Theory and Practice of Model Transformations*, pages 46–60, 2008. ISBN 978-3-540-69926-2. doi:10.1007/978-3-540-69927-9_4.

[100] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. On the evolution of OCL for capturing structural constraints in modelling languages. In *Rigorous Methods for Software Construction and Analysis*, pages 204–218, 2009. doi:10.1007/978-3-642-11447-2_13.

[101] Dimitrios S. Kolovos, Nicholas Drivalos Matragkas, Ioannis Korkontzelos, Sophia Ananiadou, and Richard F. Paige. Assessing the Use of Eclipse MDE Technologies in Open-Source Software Projects. In *Proceedings of the International Workshop on Open Source Software for Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), CEUR Workshop Proceedings, Vol. 1541*, pages 20–29, 2015.

[102] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. ISBN 0321197704.

[103] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, 44(1):117–130, Jun 2015. ISSN 1573-7462. doi:10.1007/s10462-013-9406-y.

[104] S. Lepreux, M. Abed, and C. Kolski. A human-centred methodology applied to decision support system design and evaluation in a railway network context. *Cognition, Technology & Work*, 5(4):248–271, Dec 2003. ISSN 1435-5566. doi:10.1007/s10111-003-0128-9.

[105] Lihua Li, Hong Tang, Zuobao Wu, Jianli Gong, Michael Gruidl, Jun Zou, Melvyn Tockman, and Robert A. Clark. Data Mining Techniques for Cancer Detection Using Serum Proteomic Profiling. *Artificial Intelligence in Medicine*, 32(2):71 – 83, 2004. ISSN 0933-3657. doi:10.1016/j.artmed.2004.03.006.

[106] Shih-Wei Lin and Shih-Chieh Chen. Parameter Determination and Feature Selection for C4.5 Algorithm Using Scatter Search Approach. *Soft Computing*, 16(1):63–75, Jan 2012. ISSN 1433-7479. doi:10.1007/s00500-011-0734-z.

[107] Qi Lu, Zhi-jun Lyu, Qian Xiang, Yaqin Zhou, and Jinsong Bao. Research on data mining service and its application case in complex industrial process. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1124–1129. IEEE, aug 2017. ISBN 978-1-5090-6781-7. doi:10.1109/COASE.2017.8256255.

[108] T D Luu, A Rusu, V Walter, B Linard, L Poidevin, R Ripp, L Moulinier, J Muller, W Raffelsberger, N Wicker, O Lecompte, J D Thompson, O Poch, and H Nguyen. KD4v: comprehensible knowledge discovery system for missense variant. *Nucleic Acids Research*, 40:W71–W75, 2012. ISSN 0305-1048. doi:10.1093/nar/gks474.

[109] M. Ochodek et al. Using Machine Learning to Design a Flexible LOC Counter. In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pages 14–20, Feb 2017. doi:10.1109/MALTESQUE.2017.7882011.

[110] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[111] Ruchika Malhotra. A Systematic Review of Machine Learning Techniques for Software Fault Prediction. *Applied Soft Computing*, 27, 2015. ISSN 1568-4946. doi:10.1016/j.asoc.2014.11.023.

[112] Elzbieta Malinowski and Esteban Zimanyi. *Advanced Data Warehouse Design, From Conventional to Spatial and Temporal Application*. Springer, 2008. ISBN 9783540744047.

[113] Yulkeidi Martínez, Cristina Cachero, and Santiago Meliá. Evaluating the Impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams. In Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf, editors, *Web Engineering*, pages 223–237, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31753-8. doi:10.1007/978-3-642-31753-8_17.

[114] Wes McKinney. Data Structures for Statistical Computing in Python . In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[115] David A Mellis, Ben Zhang, Audrey Leung, and Björn Hartmann. Machine Learning for Makers. In *Proceedings of the 2017 Conference on Designing Interactive Systems - DIS '17*, volume 2, pages 1213–1225. ACM Press, 2017. ISBN 9781450349222. doi:10.1145/3064663.3064735.

[116] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005. ISSN 03600300. doi:10.1145/1118890.1118892.

[117] Rolf Molich and Jakob Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3):338–348, March 1990. ISSN 0001-0782. doi:10.1145/77481.77486.

[118] Martin Monperrus, Jean Marc Jézéquel, Benoit Baudry, Joël Champeau, and Brigitte Hoeltzener. Model-driven generative development of measurement software. *Software and Systems Modeling*, 10(4):537–552, 2011. ISSN 16191366. doi:10.1007/s10270-010-0165-9.

[119] M. Arthur Munson. A study on the importance of and time spent on different modeling steps. *SIGKDD Explor. Newsl.*, 13(2):65–71, May 2012. ISSN 1931-0145. doi:10.1145/2207243.2207253.

[120] Carlos Márquez-Vera, Alberto Cano, Cristobal Romero, Amin Yousef Mohammad Noaman, Habib Mousa Fardoun, and Sebastian Ventura. Early dropout prediction using data mining: a case study with high school students. *Expert Systems*, 33 (1):107–124, 2016. doi:10.1111/exsy.12135.

[121] Daniel Müllner. fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. *Journal of Statistical Software, Articles*, 53(9):1–18, 2013. ISSN 1548-7660. doi:10.18637/jss.v053.i09.

[122] Solomon Negash. Business intelligence. *Communications of the Association for Information Systems*, 13(15), 2004.

[123] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.

[124] Object Management Group. Object Constraint Language Specification (OCL). https://www.omg.org/spec/OCL/, 2014.

[125] Object Management Group. Meta Object Facility (MOF) Core Specification. https://www.omg.org/spec/MOF/, 2016.

[126] Object Management Group. Unified Modeling Language (UML) Specification. https://www.omg.org/spec/UML/, 2017.

[127] Wienand A. Omta, Roy G. van Heesbeen, Romina J. Pagliero, Lieke M. van der Velden, Daphne Lelieveld, Mehdi Nellen, Maik Kramer, Marley Yeong, Amir M. Saeidi, Rene H. Medema, Marco Spruit, Sjaak Brinkkemper, Judith Klumperman, and David A. Egan. HC StratoMineR: A Web-Based Tool for the Rapid Analysis of High-Content Datasets. *ASSAY and Drug Development Technologies*, 14(8): adt.2016.726, 2016. ISSN 1540-658X. doi:10.1089/adt.2016.726.

[128] Richard F. Paige, Dimitrios S. Kolovos, Louis M. Rose, Nicholas Drivalos, and Fiona A. C. . The Design of a Conceptual Framework and Technical Infrastructure for Model Management Language Engineering. In *IEEE International Conference on Engineering of Complex Computer Systems*, 2009. ISBN 978-0-7695-3702-3. doi:10.1109/ICECCS.2009.14.

[129] Richard F. Paige, Dimitrios S. Kolovos, and Fiona A C Polack. A tutorial on metamodelling for grammar researchers. *Science of Computer Programming*, 96 (P4):396–416, 2014. ISSN 01676423. doi:10.1016/j.scico.2014.05.007.

[130] Terence Parr, Sam Harwell, and Kathleen Fisher. Adaptive ll(*) parsing: The power of dynamic analysis. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 579–598, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2585-1. doi:10.1145/2660193.2660202.

[131] Dau Pelleg and Andrew Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the 17th International Conference on Machine Learning*, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2.

[132] C Peng, M Deng, and L Di. User-oriented agricultural drought information cluster. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3105–3108, 2014. ISBN 9781479957750 (ISBN). doi:10.1109/IGARSS.2014.6947134.

[133] Chunming Peng, Meixia Deng, Liping Di, and Weiguo Han. Delivery of agricultural drought information via web services. *Earth Science Informatics*, 8(3): 527–538, 2015. ISSN 1865-0473. doi:10.1007/s12145-014-0198-7.

[134] Carla Proietti, Martha Zakrzewski, Thomas S. Watkins, Bernard Berger, Shihab Hasan, Champa N. Ratnatunga, Marie-Jo Brion, Peter D. Crompton, John J. Miles, Denise L. Doolan, and Lutz Krause. Mining, visualizing and comparing multidimensional biomolecular data using the Genomics Data Miner (GMine) Web-Server. *Scientific Reports*, 6(November):38178, 2016. ISSN 2045-2322. doi:10.1038/srep38178.

[135] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[136] R. Hebig et al. The Quest for Open Source Projects That Use UML: Mining GitHub. In *ACM/IEEE 19th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, pages 173–183, 2016. ISBN 978-1-4503-4321-3. doi:10.1145/2976767.2976778.

[137] Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, 2014. ISSN 1433-7541. doi:10.1007/s10044-012-0280-z.

[138] William Rice. *Moodle 2.0 E-Learning Course Development*. Packt Publishing, 2011. ISBN 9781849515269.

[139] William Rice. *Blackboard Essentials for Teachers*. Packt Publishing, July 2012. ISBN 9781849692922.

[140] Cristobal Romero and Sebastian Ventura. Data mining in education. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(1):12–27, 2013. doi:10.1002/widm.1075.

[141] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona A. Polack. The Epsilon Generation Language. In *Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications*, pages 1–16. Springer-Verlag, 2008. ISBN 978-3-540-69095-5. doi:10.1007/978-3-540-69100-6_1.

[142] Michele Samorani. Automatically Generate a Flat Mining Table with Dataconda. *Proceedings - 15th IEEE International Conference on Data Mining Workshop, ICDMW 2015*, (Figure 1):1644–1647, 2016. doi:10.1109/ICDMW.2015.100.

[143] A Santos and R Camacho. ILP made easy C3. In *IADIS European Conference on Data Mining, Part of the IADIS Multi Conference on Computer Science and Information Systems, MCCSIS*, pages 175–180, 2011. ISBN 9789728939533 (ISBN).

[144] Adrian Santos, Janne Järvinen, Jari Partanen, Markku Oivo, and Natalia Juristo. Does the performance of TDD hold across software companies and premises? A group of industrial experiments on TDD. In *Product-Focused Software Process Improvement - 19th International Conference, PROFES*, pages 227–242, 2018. doi:10.1007/978-3-030-03673-7_17.

[145] R S Santos, S M F Malheiros, S Cavalheiro, and J M P de Oliveira. A data mining system for providing analytical information on brain tumors to public health decision makers. *Computer Methods and Programs in Biomedicine*, 109: 269–282, 2013. ISSN 01692607. doi:10.1016/j.cmpb.2012.10.010.

[146] Peggy Schlesinger and Nayem Rahman. Self-Service Business Intelligence Resulting in Disruptive Technology. *Journal of Computer Information Systems*, 56(1): 11–21, 2016. doi:10.1080/08874417.2015.11645796.

[147] David Schuff, Karen Corral, Robert D. St. Louis, and Greg Schymik. Enabling self-service BI: A methodology and a case study for a model management warehouse. *Information Systems Frontiers*, 20(2):275–288, 2018. ISSN 15729419. doi:10.1007/s10796-016-9722-2.

[148] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5): 19–25, 2003. doi:10.1109/MS.2003.1231146.

[149] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Comput. Surv.*, 45(3): 31:1–31:35, July 2013. ISSN 0360-0300. doi:10.1145/2480741.2480748.

[150] Jack W Smith and et al. Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265, November 1988. ISSN 0195-4210.

[151] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.

[152] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: clipse Modeling Framework*. Addison-Wesley Professional, 2nd edition, 2009. ISBN 0321331885.

[153] Safwan Sulaiman, Tariq Mahmoud, Stephan Robbers, Jorge Marx Gómez, and Joachim Kurzhöfer. A Tracing System for User Interactions towards Knowledge Extraction of Power Users in Business Intelligence Systems. In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, volume 3, pages 199–207, 2016. ISBN 978-989-758-203-5. doi:10.5220/0006053601990207.

[154] Mark Sweney. Netflix gathers detailed viewer data to guide its search for the next hit. The Guardian, 2014. URL https://goo.gl/4nQVxE. last accessed: March 2019.

[155] Eugene Syriani, Lechanceux Luhunu, and Houari Sahraoui. Systematic mapping study of template-based code generation. *Computer Languages, Systems and Structures*, 52:43–62, 2018. ISSN 14778424. doi:10.1016/j.cl.2017.11.003.

[156] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. Automated parameter optimization of classification techniques for defect prediction models. In *38th International Conference on Software Engineering (ICSE)*, pages 321–332, 2016. ISBN 978-1-4503-3900-1. doi:10.1145/2884781.2884857.

[157] The R Project for Statistical Computing. https://www.r-project.org/.

[158] K. Trase and E. Fink. A model-driven visualization tool for use with model-based systems engineering projects. In *2014 IEEE Aerospace Conference*, pages 1–10, March 2014. doi:10.1109/AERO.2014.6836268.

[159] U. Fayyad et al. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Commun. ACM*, 39:27–34, November 1996. ISSN 0001-0782. doi:10.1145/240455.240464.

[160] Gitte Vanwinckelen and Hendrik Blockeel. A declarative query language for statistical inference. *ECML/PKDD 2013 Workshop: Languages for Data Mining and Machine Learning*, 2013.

[161] Matias Ezequiel Vara Larsen and Arda Goknil. Railroad Crossing Heterogeneous Model. In *GEMOC workshop 2013 - International Workshop on The Globalization of Modeling Languages*, Miami, Florida, United States, September 2013. URL https://hal.inria.fr/hal-00867316.

[162] Markus Voelter, Bernd Kolb, Tamás Szabó, Daniel Ratiu, and Arie van Deursen. Lessons learned from developing mbeddr: a case study in language engineering with MPS. *Software and System Modeling*, 18(1):585–630, 2019. doi:10.1007/s10270-016-0575-4.

[163] Markus Völter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart C. L. Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013. ISBN 978-1-4812-1858-0.

[164] Jane Webster and Richard T Watson. Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly*, 26(2):13–23, 2002.

[165] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(1):1–20, 2007. ISSN 1548-7660. doi:10.18637/jss.v021.i12.

[166] Rüdiger Wirth. CRISP-DM: Towards a Standard Process Model for Data Mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, 2000.

[167] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016. ISBN 0128042915, 9780128042915.

[168] Claes Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10, May 2014. doi:10.1145/2601248.2601268.

[169] Claes Wohlin and Rafael Prikladnicki. Systematic Literature Reviews in Software Engineering. *Information and Software Technology*, 55(6):919–920, jun 2013. doi:10.1016/j.infsof.2013.02.002.

[170] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012. doi:10.1007/978-3-642-29044-2.

[171] D. H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, Oct 1996. ISSN 0899-7667. doi:10.1162/neco.1996.8.7.1341.

[172] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997. ISSN 1089-778X. doi:10.1109/4235.585893.

[173] Robert Wrembel and Christian Koncilia. *Data Warehouses And Olap: Concepts, Architectures And Solutions*. IRM Press, 2006.

[174] Yelp. Yelp Dataset Challenge Round 9. https://www.yelp.com/dataset_challenge. [Online; accessed 21-May-2019].

[175] Athanasios Zolotas, Horacio Hoyos Rodriguez, Dimitrios S. Kolovos, Richard F. Paige, and Stuart Hutchesson. Bridging Proprietary Modelling and Open-Source Model Management Tools: The Case of PTC Integrity Modeller and Epsilon. *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 237–247, 2017. doi:10.1109/MODELS.2017.18.

[176] M Zorrilla and D García-Saiz. A Service Oriented Architecture to Provide Data Mining Services for Non-Expert Data Miners. *Decision Support Systems*, 55: 399–411, 2013. ISSN 01679236. doi:10.1016/j.dss.2012.05.045.

[177] Marta E. Zorrilla, Diego García-saiz, and Jose L. Balcázar. Towards parameter-free data mining: Mining educational data with yacaree. In *EDM 2011 - Proceedings of the 4th International Conference on Educational Data Mining*, pages 363–364, 2011.

[178] Ines Čeh, Matej Črepinšek, Tomaž Kosar, and Marjan Mernik. Ontology driven development of domain-specific languages. *Computer Science and Information Systems*, 8(2):317–342, 2011. ISSN 1820-0214. doi:10.2298/CSIS101231019C.

Appendix A

Comments on Ad-Hoc Applications

As previously commented in the selection strategy for scientific databases (see Section 2.1.4), we excluded a special category of papers from our evaluation process. This category is composed of *ad-hoc applications*, which were developed to solve a concrete analysis problem on a specific domain. Proof of this is that, for instance, we found several applications focusing on the analysis of large, multidimensional datasets [92, 127, 134]; on applying Inductive Logic Programming [108, 143]; or on analysing time-series data [34, 67, 115]. However, each application was developed independently, and adapted to a concrete domain (e.g medicine for [92], biomedicine for [127], and biomolecular for [134]). This kind of conventional, personalised development is the one that escaped from the focus of this review.

Nevertheless, we decided to include a short summary of these applications, because they made a special effort to be usable by domain experts without technical knowledge in data mining techniques. We considered that this effort could be transferred to other applications, so they are of interest for the ultimate goal of democratising data mining.

Applications of this category offer solutions for very concrete problems. These applications target heterogeneous domains, including molecular biology, medicine, biomedicine, genomics, education, electronics and agriculture. Table A.1 shows the domain of each application, and the specific help they provide for non-experts and that we consider relevant for the general purpose of data mining democratisation.

For instance, Chittaro et al. [34] applies time-series analysis methods for the monitoring of haemodialysis processes. This method is integrated in a user-friendly system, which was developed by data mining experts. The system allows clinicians to envision the evolution over time of different metrics presented in 2D or 3D user-friendly visualizations. The application includes a special circular control panel, which allows clinicians to tune the visualization and analysis parameters. Despite being an application focused on the study of haemodialysis, part of this solution could be transferred to the visualization and analysis of time series data coming from other domains.

Table A.1 Characteristics of encountered ad-hoc applications for non-experts.

Reference	Domain	Contribution to data mining democratisation
Chittaro et al. [34]	Medicine	A circular control panel for the management of 2D and 3D visualisations of time series data.
García et al. [64]	Education	An association rule learning tool for the analysis of educational data that allows sharing the obtained rules between courses of similar nature.
Guyet et al. [67]	Medicine	A tool to perform pattern matching visually in time series data (e.g. respiratory data).
Jugo et al. [84]	Education	Web-based interface for clustering analysis and for guiding users in a learning path.
Kamdar et al. [85]	Biomedicine	A high-level system to formulate data queries over ontology-based data sources.
Kamsu-Foguem et al. [86]	Medicine	A machine learning-enhanced monitor that automatically determines the most relevant indicators to show in a constrained-size display (e.g. medical intensive care units).
Klenk et al. [92]	Medicine	A system to perform case-based reasoning (e.g. history-based survival analysis of a patient).
Luu et al. [108]	Genomics	An Inductive Logic Programming (ILP) system that analyses data and gives rule-based explanations understandable by non-experts in data mining.
Peng et al. [133]	Agriculture	A geographic system that allows obtaining map-based visualisations of different indicators of interest (e.g. drought maps of certain areas)
Mellis et al. [115]	Electronics	A high-level solution for the analysis of raw sensor data.
Omta et al. [127]	Biomedicine	Web service for analysing High-Content (i.e. large and multidimensional) datasets.
Proietti et al. [134]	Biomolecular	Web service for the analysis and comparison of different multidimensional datasets.
Santos and Camacho [143]	Biomolecular	An ILP system for multi-relational data mining (similar objectives to Luu et al. [108]).

Appendix B

Class Diagrams Dataset Extractions: Pinset vs. ETL

The following sections show how certain dataset extractions would be performed using either Pinset or ETL. These extractions take place over UML class diagrams, as explained in Section 5.5. The set of performed extractions seeks to analyse the benefits offered by Pinset's specific column generators when compared with using a general purpose transformation language, where these generators are not available.

Because of formatting, it is possible that the scripts of this appendix does not return the exact same measurements as provided in Table 5.4 of the comparison (see Section 5.6.4). The original extractions can be found in an external repository[1][2][3][4].

Operations Used By ETL Scripts

The following operations are required in all ETL examples. These operations were not taken into account when analysing script size in the comparison of Section 5.6.4.

Listing B.1 Operations Used by ETL scripts.

```
1  operation createDataset(datasetName: String): Dataset!Dataset {
2    var d = new Dataset!Dataset();
3    d.name = datasetName;
4    return d;
5  }
6
7  operation createColumn(colName: String): Dataset!Column {
8    var col = new Dataset!Column();
9    col.name = colName;
10   return col;
11 }
12
13 operation createCell(col: Dataset!Column, val: Any): Dataset!Cell {
14   var cell = new Dataset!Cell();
15   cell.column = col;
16   cell.value = val;
17   return cell;
18 }
```

[1] https://github.com/alfonsodelavega/pinset-examples/blob/master/es.unican.istr.pinset.examples/01-examples.pset

[2] https://github.com/alfonsodelavega/pinset-examples/blob/master/es.unican.istr.pinset.examples/02-allMetrics.pset

[3] https://github.com/alfonsodelavega/pinset-examples/blob/master/es.unican.istr.pinset.examples.etlComparison/etl/01-examples.etl

[4] https://github.com/alfonsodelavega/pinset-examples/blob/master/es.unican.istr.pinset.examples.etlComparison/etl/02-allMetrics.etl

B.1 Basic Class Metrics

Listing B.2 Basic Class Metrics (ETL).

```
pre {
  var modelRoot = new Dataset!Model();
  var metricsDataset = createDataset("BasicMetrics");
  modelRoot.datasets.add(metricsDataset);
  var bmd_c_name = createColumn("name");
  var bmd_c_isAbstract = createColumn("isAbstract");
  var bmd_c_parentName = createColumn("parentName");
  var bmd_c_OO_NOA = createColumn("OO_NOA");
  var bmd_c_OO_NOM = createColumn("OO_NOM");
  var bmd_c_OO_NOF = createColumn("OO_NOF");
  var bmd_c_CK_DIT = createColumn("CK_DIT");
  metricsDataset.columns =
  Collection {bmd_c_name, bmd_c_isAbstract,
    bmd_c_parentName, bmd_c_OO_NOA,
    bmd_c_OO_NOM, bmd_c_OO_NOF,
    bmd_c_CK_DIT};
}

rule BasicMetricsClass2Row
transform class : Model!Class
to row : Dataset!Row {
  metricsDataset.rows.add(row);
  row.cells.add(createCell(bmd_c_name, class.name));
  row.cells.add(createCell(bmd_c_isAbstract,
  class.isAbstract));
  var parentName = "";
  if (not class.superClass.isEmpty()) {
    parentName = class.superClass.first().name;
  }
  row.cells.add(createCell(bmd_c_parentName,
  parentName));
  var OO_NOA = class.attributes.size();
  var OO_NOM = class.operations.size();
  var OO_NOF = OO_NOA + OO_NOM;
  row.cells.add(createCell(bmd_c_OO_NOA, OO_NOA));
  row.cells.add(createCell(bmd_c_OO_NOM, OO_NOM));
  row.cells.add(createCell(bmd_c_OO_NOF, OO_NOF));
  row.cells.add(createCell(bmd_c_CK_DIT, class.dit()));
}

```

```
41 operation Class dit(): Integer {
42   var dit = 0;
43   var node = self;
44   while (not node.superClass.isEmpty()) {
45     node = node.superClass.first();
46     dit += 1;
47   }
48   return dit;
49 }
```

Listing B.3 Basic Class Metrics (Pinset).

```
1  dataset basicClassMetrics over class : Class {
2    column name : class.name
3    column isAbstract : class.isAbstract
4    column parentName {
5      var name = null;
6      if (not class.superClass.isEmpty()) {
7        name = class.superClass.first().name;
8      }
9      return name;
10   }
11   column OO_NOA : class.attributes.size()
12   column OO_NOM : class.operations.size()
13   column OO_NOF : OO_NOA + OO_NOM
14   column CK_DIT : class.dit()
15 }
16
17 operation Class dit(): Integer {
18   var dit = 0;
19   var node = self;
20   while (not node.superClass.isEmpty()) {
21     node = node.superClass.first();
22     dit += 1;
23   }
24   return dit;
25 }
```

B.2 Features Accesors

Listing B.4 Features Accessors (ETL).

```
pre classBasicInfoPre {
  var modelRoot = new Dataset!Model();
  var classBasicInfoDataset = createDataset("classBasicInfo");
  modelRoot.datasets.add(classBasicInfoDataset);
  var cbi_c_name = createColumn("name");
  var cbi_c_isAbstract = createColumn("isAbstract");
  var cbi_c_packageName = createColumn("package_name");
  classBasicInfoDataset.columns =
    Sequence {cbi_c_name, cbi_c_isAbstract, cbi_c_packageName};
}
@greedy
rule ClassBasicInfo
transform class : Model!Class
to row : Dataset!Row {
  classBasicInfoDataset.rows.add(row);
  row.cells.add(createCell(cbi_c_name, class.name));
  row.cells.add(createCell(cbi_c_isAbstract, class.isAbstract));
  var packageName = null;
  if (class.package <> null) {packageName = class.package.name;}
  row.cells.add(createCell(cbi_c_packageName, packageName));
}
```

Listing B.5 Features Accessors (Pinset).

```
dataset classBasicInfo over class : Model!Class {
  properties [name, isAbstract]
  reference package[name]
}
```

B.3 Extended Accessors

Listing B.6 Extended Features Accessors (ETL).

```
pre classBasicInfoExtendedPre {
  var cbiExtendedDataset = createDataset("classBasicInfoExtended");
  modelRoot.datasets.add(cbiExtendedDataset);
  var cbie_c_name = createColumn("name");
  var cbie_c_isAbstract = createColumn("isAbstract");
  var cbie_c_isLeaf = createColumn("isLeaf");
  var cbie_c_qualifiedName = createColumn("qualifiedName");
```

```
8    var cbie_c_visibility = createColumn("visibility");
9    var cbie_c_packageName = createColumn("package_name");
10   var cbie_c_packageQualifiedName =
11     createColumn("package_qualifiedName");
12   cbiExtendedDataset.columns = Sequence {cbie_c_name,
13     cbie_c_isAbstract, cbie_c_isLeaf, cbie_c_qualifiedName,
14     cbie_c_visibility, cbie_c_packageName,
15     cbie_c_packageQualifiedName};
16 }
17 @greedy
18 rule ClassBasicInfoExtended
19 transform class : Model!Class
20 to row : Dataset!Row {
21   cbiExtendedDataset.rows.add(row);
22   row.cells.add(createCell(cbie_c_name, class.name));
23   row.cells.add(createCell(cbie_c_isAbstract, class.isAbstract));
24   row.cells.add(createCell(cbie_c_isLeaf, class.isLeaf));
25   row.cells.add(createCell(cbie_c_qualifiedName,
26                            class.qualifiedName));
27   row.cells.add(createCell(cbie_c_visibility, class.visibility));
28   var packageName = null;
29   if (class.package <> null) {packageName = class.package.name;}
30   row.cells.add(createCell(cbie_c_packageName, packageName));
31   var packageQualifiedName = "";
32   if (class.package <> null) {
33     packageQualifiedName = class.package.qualifiedName;
34   }
35   row.cells.add(
36     createCell(cbie_c_packageQualifiedName, packageQualifiedName));
37 }
```

Listing B.7 Extended Features Accessors (Pinset).

```
1 dataset classBasicInfoExtended over class : Model!Class {
2   properties [name, isAbstract, isLeaf, qualifiedName, visibility]
3   reference package[name, qualifiedName]
4 }
```

B.4 Filtering

Listing B.8 Filtering Examples (ETL).

```
pre classSelectionPre {
  var csDataset = createDataset("classSelection");
  modelRoot.datasets.add(csDataset);
  var cs_c_name = createColumn("name");
  csDataset.columns = Sequence {cs_c_name};
}
@greedy
rule ClassSelection
transform class : Model!Class
to row : Dataset!Row {
  guard : class.isAbstract
  csDataset.rows.add(row);
  row.cells.add(createCell(cs_c_name, class.name));
}
```

Listing B.9 Filtering Examples, two versions (Pinset).

```
dataset classSelectionGuard over class : Model!Class {
  guard : class.isAbstract
  properties [name]
}

dataset classSelectionFrom over class : Model!Class
from : Model!Class.all().select(c | c.isAbstract) {
  properties [name]
}
```

B.5 Grid

Listing B.10 Grid Example (ETL).

```
pre attributesByVisibilityPre {
  var visDataset = createDataset("attributesByVisibility");
  modelRoot.datasets.add(visDataset);
  var cs_c_name = createColumn("name");
  visDataset.columns = Sequence {cs_c_name};
  var visibilities = Sequence{UML!VisibilityKind#public,
    UML!VisibilityKind#protected,
    UML!VisibilityKind#private,
    UML!VisibilityKind#package};
```

```
10    var visibilityColumns = Map{};
11    for (visibility in visibilities) {
12      var columnName = "#" + visibility + "_attrs";
13      var column = createColumn(columnName);
14      visibilityColumns.put(columnName, column);
15      visDataset.columns.add(column);
16    }
17  }
18  @greedy
19  rule AttributesByVisibility
20  transform class : Model!Class
21  to row : Dataset!Row {
22    visDataset.rows.add(row);
23    row.cells.add(createCell(cbi_c_name, class.name));
24    for (visibility in visibilities) {
25      var column = visibilityColumns.get("#" + visibility + "_attrs");
26      row.cells.add(createCell(column, class.attributes
27      .select(a | a.visibility = visibility)
28      .size()));
29    }
30  }
```

Listing B.11 Grid Example (Pinset).

```
1  dataset attributesByVisibility over class : Model!Class {
2    properties [name]
3    grid {
4      keys : Sequence{UML!VisibilityKind#public,
5        UML!VisibilityKind#protected,
6        UML!VisibilityKind#private,
7        UML!VisibilityKind#package}
8      header : "#" + key + "_attrs"
9      body : class.attributes
10                .select(a | a.visibility = key)
11                .size()
12    }
13  }
```

B.6 Nested From

Listing B.12 Nested From Example (ETL).

```
pre nestedFromPre {
  var nfDataset = createDataset("nestedFrom");
  modelRoot.datasets.add(nfDataset);
  var nf_c_name = createColumn("name");
  var nf_c_packageName = createColumn("package_name");
  var nf_c_packageNumClasses = createColumn("
     package_types_numClasses");
  nfDataset.columns = Sequence {nf_c_name, nf_c_packageName,
     nf_c_packageNumClasses};
}
@greedy
rule nestedFrom
transform class : Model!Class
to row : Dataset!Row {
  nfDataset.rows.add(row);
  row.cells.add(createCell(nf_c_name, class.name));
  var packageName = null;
  if (class.package <> null) {packageName = class.package.name;}
  row.cells.add(createCell(nf_c_packageName, packageName));
  var packageNumClasses = null;
  if (class.package <> null) {
    packageNumClasses = class.package.ownedType
                              .select(t | t.isKindOf(Model!Class))
                              .size();
  }
  row.cells.add(createCell(nf_c_packageNumClasses,
                           packageNumClasses));
}
```

Listing B.13 Nested From Example (Pinset).

```
dataset nestedFrom over class : Model!Class {
  properties [name]
  from package : class.package {
    properties[name]
    from types : package.ownedType {
      column numClasses : types.select(t | t.isKindOf(Model!Class))
                               .size()
    }
  }
}
```

B.7 Typeless Rules

Listing B.14 Typeles Rules (ETL).

```
pre typelessPre {
  var thMetricsDataset = createDataset("thresholdMetrics");
  modelRoot.datasets.add(thMetricsDataset);
  var thm_c_threshold = createColumn("threshold");
  var thm_c_noa = createColumn("classes_with_NOA_leq_th");
  var thm_c_nom = createColumn("classes_with_NOM_leq_th");
  var thm_c_fanIn = createColumn("classes_with_FanIn_geq_th");
  var thm_c_fanOut = createColumn("classes_with_FanOut_geq_th");
  thMetricsDataset.columns = Sequence {thm_c_threshold, thm_c_noa,
    thm_c_nom, thm_c_fanIn, thm_c_fanOut};
}

post typelessDataset {
  var allClasses = Model!Class.all();
  var thresholds = Sequence{0,1,2,5,10};
  for (threshold in thresholds) {
    var row = new Dataset!Row;
    thMetricsDataset.rows.add(row);
    row.cells.add(createCell(thm_c_threshold, threshold));
    row.cells.add(createCell(thm_c_noa,
    allClasses.select(c | c.attributes.size() <= threshold).size()))
        ;
    row.cells.add(createCell(thm_c_nom,
    allClasses.select(c | c.operations.size() <= threshold).size()))
        ;
    row.cells.add(createCell(thm_c_fanIn,
    allClasses.select(c | c.fanIn().size() >= threshold).size()));
    row.cells.add(createCell(thm_c_fanOut,
    allClasses.select(c | c.fanOut().size() >= threshold).size()));
  }
}
```

Listing B.15 Typeless Dataset Rules (Pinset).

```
pre {
  var allClasses = Model!Class.all();
}

dataset thresholdMetrics over threshold
from : Sequence{0,1,2,5,10} {
  column threshold : threshold
```

```
8    column classes_with_NOA_leq_th : allClasses.select
9      (c |c.attributes.size() <= threshold).size()
10   column classes_with_NOM_leq_th : allClasses.select
11     (c | c.operations.size() <= threshold).size()
12   column classes_with_FanIn_geq_th : allClasses.select
13     (c | c.fanIn().size() >= threshold).size()
14   column classes_with_FanOut_geq_th : allClasses.select
15     (c | c.fanOut().size() >= threshold).size()
16 }
```

B.8 All Metrics

Listing B.16 All Class Metrics (ETL).

```
1  pre {
2    var modelRoot = new Dataset!Model();
3    var allMetricsDataset = createDataset("02-allMetricsETL");
4    modelRoot.datasets.add(allMetricsDataset);
5    var c_name = createColumn("name");
6    // CK metrics
7    var c_CK_WMC = createColumn("CK_WMC");
8    var c_CK_DIT = createColumn("CK_DIT");
9    var c_CK_NOC = createColumn("CK_NOC");
10   var c_CK_CBO = createColumn("CK_CBO");
11   // OO metrics
12   var c_OO_FanIn = createColumn("OO_FanIn");
13   var c_OO_FanOut = createColumn("OO_FanOut");
14   var c_OO_NOF = createColumn("OO_NOF");
15   var c_OO_NOA = createColumn("OO_NOA");
16   var c_OO_NOPA = createColumn("NOPA");
17   var c_OO_NOPRA = createColumn("NOPRA");
18   var c_OO_NOIA = createColumn("NOIA");
19   var c_OO_NOM = createColumn("NOM");
20   var c_OO_NOPM = createColumn("NOPM");
21   var c_OO_NOPRM = createColumn("NOPRM");
22   var c_OO_NOIM = createColumn("NOIM");
23   allMetricsDataset.columns =
24   Sequence{c_name, c_CK_WMC, c_CK_DIT, c_CK_NOC, c_CK_CBO,
25     c_OO_FanIn, c_OO_FanOut, c_OO_NOF,
26     c_OO_NOA, c_OO_NOPA, c_OO_NOPRA, c_OO_NOIA,
27     c_OO_NOM, c_OO_NOPM, c_OO_NOPRM, c_OO_NOIM};
28 }
29
30
```

```
31 rule AllMetrics
32 transform class : Model!Class
33 to row : Dataset!Row {
34   allMetricsDataset.rows.add(row);
35   row.cells.add(createCell(c_name, class.name));
36   row.cells.add(createCell(c_CK_WMC, class.operations.size()));
37   row.cells.add(createCell(c_CK_DIT, class.dit()));
38   row.cells.add(createCell(c_CK_NOC, class.noc()));
39   row.cells.add(createCell(c_CK_CBO, class.cbo()));
40   row.cells.add(createCell(c_OO_FanIn, class.fanIn().size()));
41   row.cells.add(createCell(c_OO_FanOut, class.fanOut().size()));
42   row.cells.add(createCell(c_OO_NOF, class.features.size()));
43   row.cells.add(createCell(c_OO_NOA, class.attributes.size()));
44   row.cells.add(createCell(c_OO_NOPA,
45   class.attributes.select(a | a.visibility = UML!VisibilityKind#
       public).size()));
46   row.cells.add(createCell(c_OO_NOPRA,
47   class.attributes.select(a | a.visibility = UML!VisibilityKind#
       private).size()));
48   row.cells.add(createCell(c_OO_NOIA, class.allAttributes().size() -
        class.attributes.size()));
49   row.cells.add(createCell(c_OO_NOM, class.operations.size()));
50   row.cells.add(createCell(c_OO_NOPM,
51   class.operations.select(o | o.visibility = UML!VisibilityKind#
       public).size()));
52   row.cells.add(createCell(c_OO_NOPRM,
53   class.operations.select(o | o.visibility = UML!VisibilityKind#
       private).size()));
54   row.cells.add(createCell(c_OO_NOIM, class.allOperations.size -
55   class.operations.size()));
56 }
```

Listing B.17 All Class Metrics (Pinset).

```
1 dataset allClassMetrics over class : Model!Class {
2   column name : class.name
3   // CK metrics
4   column CK_WMC : class.operations.size() // simplified due to
       weight = 1
5   column CK_DIT : class.dit()
6   column CK_NOC : class.noc()
7   column CK_CBO : class.cbo()
8   // OO metrics
9   column OO_FanIn : class.fanIn().size()
```

```
10    column OO_FanOut : class.fanOut().size()
11    column OO_NOF : class.features.size()
12    column OO_NOA : class.`attributes`.size()
13    column OO_NOPA : class.`attributes`
14                          .select(a | a.visibility = UML!
                               VisibilityKind#public)
15                          .size()
16    column OO_NOPRA : class.`attributes`
17                           .select(a | a.visibility = UML!
                                VisibilityKind#private)
18                           .size()
19    column OO_NOIA : (class.allAttributes().size() -
20                      class.`attributes`.size())
21    column OO_NOM : class.operations.size()
22    column OO_NOPM : class.operations
23                          .select(o | o.visibility = UML!
                               VisibilityKind#public)
24                          .size()
25    column OO_NOPRM : class.operations
26                           .select(o | o.visibility = UML!
                                VisibilityKind#private)
27                           .size()
28    column OO_NOIM : (class.allOperations.size -
29                      class.operations.size())
30  }
```

Appendix C

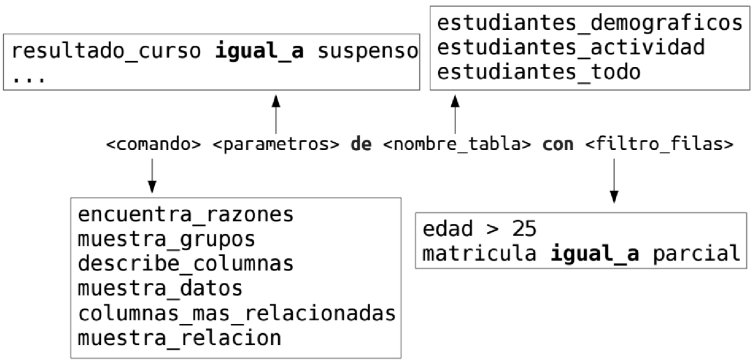
Experiments Manual (Spanish)

Manual de uso del lenguaje de análisis educacional

Este lenguaje puede utilizarse para analizar datos de los estudiantes de un curso.

Estructura de las consultas

Nota: Aunque el lenguaje esté en español, **no** deben utilizarse tildes al escribir las consultas.



Autocompletar: Pulsando **`Ctrl + Espacio`** durante la escritura de una consulta el lenguaje ofrece sugerencias para introducir comandos, nombres de tabla o de columnas, etc.

Validación: Si una consulta contiene errores, estos aparecerán en forma de un subrayado en rojo debajo del término que contiene el error, o como una cruz roja al lado de la consulta. El lenguaje ofrece mensajes explicatorios que tratan de ayudar en la detección y corrección del error.

Comandos disponibles

`encuentra_razones`	Busca causas de la ocurrencia de cierto evento: **`encuentra_razones de`** `nombre_columna` **`igual_a`** `valor` **`de`** `nombre_tabla`
`muestra_grupos`	Agrupa los datos en perfiles de acuerdo a similitudes: **`muestra_grupos de`** `nombre_tabla`
`describe_columnas`	Muestra información acerca de una tabla y sus columnas: **`describe_columnas de`** `nombre_tabla`
`muestra_datos`	Muestra los datos de una tabla: **`muestra_datos de`** `nombre_tabla`
`columnas_mas_relacionadas`	Ranking de las columnas más relacionadas con la indicada: **`columnas_mas_relacionadas con`** `nombre_columna` **`de`** `nombre_tabla`
`muestra_relacion`	Calcula la relación entre dos columnas de una tabla: **`muestra_relacion entre`** `nombre_columna_1` **`y`** `nombre_columna_2` **`de`** `nombre_tabla`

Tablas de datos educacionales

estudiantes_actividad: Datos de los estudiantes extraidos del curso online (Moodle).

id: Id del estudiante

tiempo_total: Tiempo total invertido en la plataforma (en minutos)

sesiones_totales: Número de sesiones totales en la plataforma durante el curso

duracion_media_sesion: Duración media en minutos de cada sesión

sesiones_pre_examen: Número de sesiones realizadas en la semana previa al examen

foro_visitas: Número de visitas al foro

foro_mensajes: Número de mensajes escritos en el foro

resultado_curso: Resultado del curso

Posibles valores:
- suspenso: El estudiante no ha superado el curso
- aprobado: El estudiante ha superado el curso

estudiantes_demograficos: Contiene información demográfica de los estudiantes**.**

id: Id del estudiante

edad: Edad del estudiante

estudios_previos: Máximos estudios previos terminados

Posibles valores:
- formacion_profesional: Módulo de formación profesional
- bachillerato: Estudios de bachillerato
- grado: Estudios universitarios
- universidad_inacabado: Estudios universitarios no completados

repetidor: El estudiante suspendió anteriormente este curso

Posibles valores:
- no: No es alumno repetidor
- si: Es alumno repetidor

matricula: Tipo de matrícula del estudiante

Posibles valores:
- tiempo_completo: Estudiante a tiempo completo
- tiempo_parcial: Estudiante a tiempo parcial

asiste_a_clases: El estudiante es asiduo a las clases

Posibles valores:
- no: No suele asistir a las clases
- si: Es asiduo a las clases

procedencia: Lugar de procedencia del estudiante

Posibles valores:
- nacional: El estudiante proviene de otra provincia española
- extranjero: El estudiante es extranjero
- local: El estudiante es originario de la localidad

lengua_materna: Lengua materna del estudiante

Posibles valores:
- otra: Lengua materna distinta del español
- español: Lengua materna española

resultado_curso: Resultado del curso

Posibles valores:
- suspenso: El estudiante no ha superado el curso
- aprobado: El estudiante ha superado el curso

estudiantes_todo:

Almacena las columnas demográficas y de actividad de los estudiantes.

Appendix D

Experiments Test Questions (Spanish)

FLANDM: Experimento con lenguajes de análisis de datos

1. Si deseamos comparar dos columnas de una tabla para determinar si existe alguna relación entre sus valores, ¿qué comando deberíamos emplear?

1. *Marca solo un óvalo.*

- ◯ encuentra_razones
- ◯ muestra_grupos
- ◯ describe_columnas
- ◯ muestra_datos
- ◯ columnas_mas_relacionadas
- ◯ muestra_relacion

2. ¿Cuál dirías que es el objetivo de la siguiente consulta? Descríbelo brevemente.

2.
```
muestra_datos de estudiantes_demograficos
con estudios_previos igual_a bachillerato
```

3. Ejecuta la siguiente consulta, cuyo objetivo es buscar los diferentes perfiles de estudiantes que existen en el curso en base a su actividad.

```
muestra_grupos de estudiantes_actividad
```

3. **¿Cuántos grupos o perfiles aparecen?**

4. **¿Cuál es el número medio de sesiones totales del grupo 1?**

4. Modifica la pregunta del apartado anterior, para que en este caso solamente se agrupen los estudiantes cuyo resultado del curso sea un suspenso.

5. **Copia aquí la consulta desarrollada:**

6. **¿Cuál es el número de grupos tras incluir la modificación?**

5. La siguiente consulta busca razones por las cuales los alumnos han suspendido el curso, usando para ello todos los datos disponibles. Desgraciadamente, esta consulta contiene errores en su formulación. Introdúcela en el editor y trata de corregir los errores encontrados.

```
busca_razones de resultado_curso igual_a 0
     de Estudiantes_TodosDatos
```

7. **Copia aquí la consulta corregida:**

8. **¿Cuántas reglas ha devuelto la consulta?**

6. Crea una consulta que, utilizando los datos de actividad, muestre las columnas más directamente relacionadas con el resultado del curso de los estudiantes. Escribe en esta hoja las dos primeras

columnas que te aparezcan.

9. **Copia aquí la consulta desarrollada:**

10. **¿Cuál es la columna de actividad más relacionada con el resultado del curso?**

11. **¿Cuál es la segunda columna de actividad más relacionada con el resultado del curso?**

7. Repite la pregunta anterior, utilizando ahora los datos demográficos. En este caso, escribe solamente la columna devuelta como más directamente relacionada.

12. **Copia aquí la consulta desarrollada:**

13. **¿Cuál es la columna demográfica más relacionada con el resultado del curso?**

8. Nos interesa estudiar si existe relación entre dos atributos concretos de actividad: el número total de sesiones efectuadas en moodle y el número de mensajes escritos en el foro. Escribe una consulta que calcule esta relación.

Nota: Utiliza la siguiente consulta para averiguar cuáles son los nombres concretos de las columnas que queremos analizar (o consulta el manual proporcionado):

```
describe_columnas de estudiantes_actividad
```

14. **Copia aquí la consulta desarrollada para obtener la relación entre las columnas estudiadas:**

15. **¿Cuál es la relación entre el número de sesiones en moodle y el número de mensajes escritos en el foro (valor numérico)?**