Variability-aware data migration tool

David Romero
Dept. of Computer Languages and Systems
University of Seville
Seville, Spain
drorganvidez@us.es

José A. Galindo
Dept. of Computer Languages and Systems
University of Seville
Seville, Spain
jgalindo@us.es

Jose-Miguel Horcas
Dept. of Computer Languages and Systems
University of Seville
Seville, Spain
jhorcas@us.es

David Benavides
Dept. of Computer Languages and Systems
University of Seville
Seville, Spain
benavides@us.es

ABSTRACT

Relational databases are widely present in the development of software applications. A typical implementation can be seen in content management systems found on most websites. However, the migration of database structure and content between different management systems is not trivial, and the manual creation of scripts makes it difficult to reuse them in other scenarios. This paper presents a tool for database migration by modeling what we call a migration product line. This tool allows to obtain different configurations resulting in final products in a semi-automatic way, i.e., products according to software requirements, considering the variability between any two relational databases. To study the feasibility of our proposal, we have implemented a proof of concept that performs the migration between two databases.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Requirements analysis*; *Software design engineering*; *Software implementation planning*.

KEYWORDS

databases,
migrations,
model transformations,
product lines,
variability

1 INTRODUCTION

One of the problems when information systems evolve and suffer from technical debt, is the migration of data [7]. Old data structures must be migrated to new ones, or simply a piece of a system is changed, and so are its corresponding data structures. When adding a feature to an existing product, data migration can happen too. This migration happens not only in software product lines but also in system development.

Let's take Content Management Systems (CMSs) as an illustrative example. CMSs are present on many websites due to their ease of implementation and the addition of new components that extend the use of the system [1]. Examples such as Drupal or WordPress allow speeding up the development of information systems, as there are already predefined solutions compared to more traditional development [6].

These CMSs use a relational database, typically MySQL[1]. A CMS's database structure is tied to a particular version of the software, which can be modified by adding new components or plugins. However, migrating database structure and content between two CMSs is not trivial. Although certain *plugins* allow migration of the base system, they are limited to specific versions of the source system and specific versions of the target system. Regardless of the pair of systems we want to study, an option is the manual creation of SQL scripts[2] detailing how to migrate each table and each attribute, among others. However, this solution is impractical because of the variability present in the two databases and the tight coupling with their respective CMSs. Let V_A be the number of versions of a CMS A and V_B be the number of versions of a CMS B. The number of scripts required for a CMS S to migrate a system from A to B and vice versa will be determined by $S = 2 \cdot V_A \cdot V_B$.

There are 531 current Drupal versions[3] and 574 current WordPress versions[4]. Trying to make the migration bidirectional, we get a total of 609,588 different SQL scripts. These scripts are not reusable with each other, something unfeasible in technical terms. In addition, the *scripts* would contemplate the complete migration from A to B but not small-scale migrations, e.g., migrating only certain tables. Since each version can have a variable number of tables, let T_A be the number of tables of CMS A, ATR_A the number of attributes of each table, TA_A the number of type of each attribute, and making a simile with CMS B, we have $S = 2 \cdot V_A \cdot T_A \cdot ATR_A \cdot TA_A \cdot V_B \cdot T_B \cdot ATR_B \cdot TA_B \cdot TA_B$.

Let us assume that the two CMSs have the same number of tables, e.g., 8, 5 attributes for each table along with their five types and that CMS B has the same structure except for a new table with two attributes. If we apply the formula, we are left with $S = 2 \cdot 531 \cdot 8 \cdot 5 \cdot 5 \cdot 574 \cdot (8 \cdot 5 \cdot 5 + (1 \cdot 2 \cdot 2)) = 2.49 \cdot 10^{10}$ *scripts* different for the possible migrations *partially*. We should add the presence of *plugins* that introduces their modification into each database. For practical purposes, it is impossible to manually cover all these configurations given the high variability of the problem.

[1] https://dev.mysql.com/doc
[2] https://gist.github.com/ryelle/3823356
[3] https://github.com/drupal/drupal/tags
[4] https://github.com/WordPress/WordPress/tags

Currently, there are Domain Specific Languages (DSLs) capable of modeling migrations between two artifacts [8]; in our case, two databases. However, the same problem remains, which is variability. We understand the use of a DSL of this type as a tool that helps us and makes it easier to model a migration between two databases [4], which is a way to avoid programming the SQL script directly.

In section 2, we propose a tool where the domain expert user can model a migration between two relational databases; in section 3, we show a demo, and finally, in section 4 we present conclusions and future work.

2 TOOL: DATABASE TRANSFORMATIONS USING A MIGRATION PRODUCT LINE

We propose to develop a tool that allows the modeling of migrations within a Migration Product Line through an interactive process by the domain expert user. The tool gives the user a choice of valid actions to configure their migration without having to use a DSL. In Figure 1 shows our proposal for generating such *script* from two relational databases.

2.1 Extract simplified model

To facilitate the migration and study of each database, we have devised a simplified model called *Simple Database Model* (SDM). An SDM, which can be seen in the Listing 1, is an XML file that contains a description of the entities, the relationships between them, the associated attributes, and the type (e.g., char, varchar, bool). This process can be manual, that is, the user details in the XML, the structure of the source (or target) database, or it can be extracted automatically by making a call to the machine's MySQL server. In our tool, we provide a utility that allows the automatic generation of the SDM.

Listing 1: Simple Database Model example.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<sdm>

    <entity id="post">
        <name>Post</name>
        <attribute>
            <name>title</name>
            <type>varchar(25)</type>
        </attribute>
        <attribute>
            <name>body</name>
            <type>text</type>
        </attribute>
    </entity>

    <entity id="post_meta">
        <name>PostMeta</name>
        <attribute>
            <name>author</name>
            <type>varchar(25)</type>
        </attribute>
    </entity>

    <relation>
        <one id="post"></one>
        <many id="post_meta"></many>
    </relation>

</sdm>
```

2.2 Select actions

If we have already obtained their respective SMD's from the source and target databases, the system studies their differences. Since the source database will transform until it reaches the target database, the tool gives the user the choice of a series of available actions from a predefined set of actions.

We have implemented a total of 9 predefined actions (but more can be added):

- Concerning entities: *create entity, rename entity, delete entity.*
- Concerning attributes: *create attribute, move attribute (from one entity to another), rename attribute, change attribute type, delete attribute.*
- Concerning relationships: *create relationship.*

The list of actions available in the migration is variable, i.e., conditional on the current status between the two bases. Each time the domain expert user selects an action, the tool generates a series of new possible actions. For example, if the user has selected the action *Create entity*, the tool will get new available actions such as *Create attribute* or *Move attribute* to that entity. The selection of an action can be manual or automatic. To implement automatic selection, we have used a heuristic [5], which is a numerical estimate of the difference between the two SDM.

Let X be the source base, Y the target base. Let E be an entity, At be an attribute of an entity, and T be the type of each attribute. Let $dif(A_1, B_2)$ be the nº of entities/attributes/attribute types of attributes of X that do not appear in Y and the nº of entities/attributes/attribute types of Y that do not appear in X.

$$H = dif(E_x, E_y) + \sum_{n=1}^{|E_x|} \sum_{m=1}^{|E_y|} dif(At_n, At_m) + dif(T_n, T_m)$$

For example, suppose the target database has one entity and two new attributes that do not appear in the source database. The heuristic will be at least +3 since these are the minimum actions we will perform: *Create entity* (once) and *Create attribute* (twice). A greedy algorithm [3] selects the action that most decreases this heuristic and obtains a series of valid actions when it reaches 0. Other heuristics can be added in the future.

2.3 Transform

The list of actions to be executed, obtained manually or by heuristics, generates what we have called a *Simple Transformation Model* (STM), which is an XML file detailing the type of action, the entities involved, and the minimum information to carry out the transformation. We can see an example instance in the Listing 2.

In our example, we have selected the action *Change attribute type* and the action *Rename attribute*. The selection of these two actions has led to creating two types of transformations, both of type *attribute*. Within each type of transformation there are predefined types of actions, in our case, the actions *retype* and *rename*. This structure allows the grouping of several actions of the same action type, which will consider in future implementations of the tool.

The STM is translated into SQL statements with the help of a template engine [5]. The tool default copies the structure and data from the source database to the final SQL script. Then, it applies

[5] https://jinja.palletsprojects.com/en/3.1.x

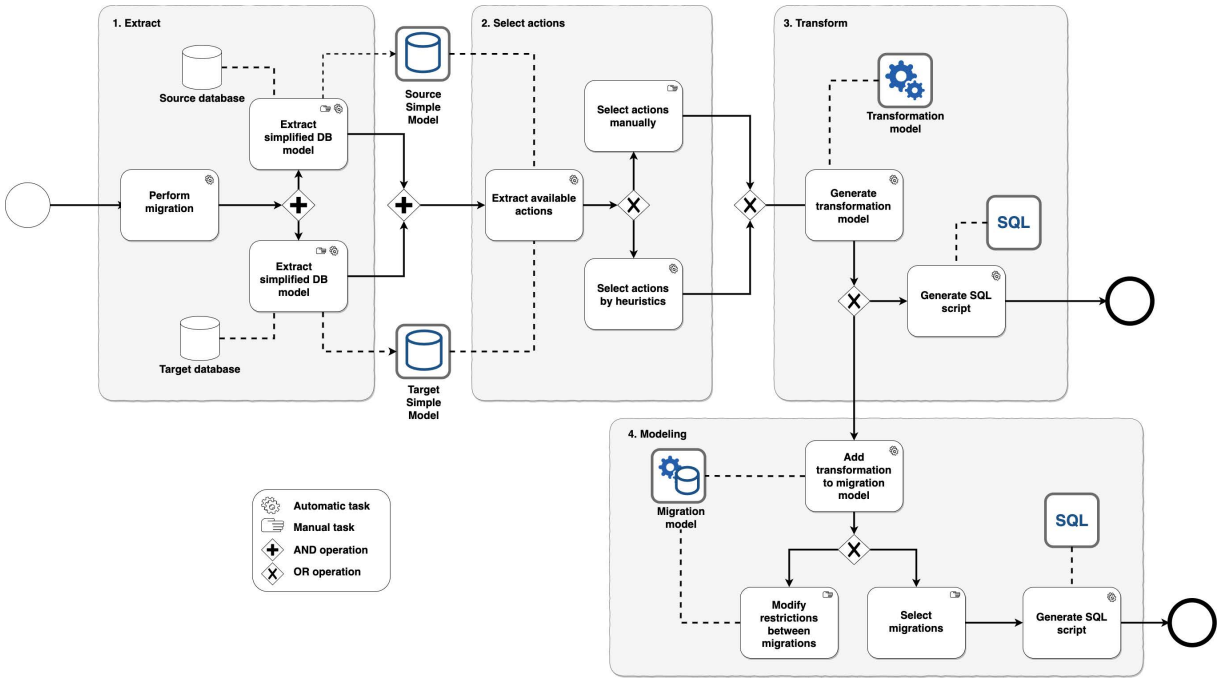

Figure 1: Description of components of the proposed tool.

the transformations that the user has defined. That is, it translates into SQL (MySQL) each transformation according to the variability. These transformations are never incompatible because the tool has been mutating the list of available actions taking into account the previous selection immediately before.

Listing 2: Simple Transformation Model example

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<stm>

    <transformation type="attribute" id="ChangeAuthorType">
        <action type="retype">
            <entity>post_meta</entity>
            <attribute>author</attribute>
            <retype>VARCHAR(100)</retype>
        </action>
    </transformation>

    <transformation type="attribute" id="RenameAuthorName">
        <action type="rename">
            <entity>post_meta</entity>
            <attribute>author</attribute>
            <rename>owner</rename>
        </action>
    </transformation>

</stm
```

2.4 Modeling into Migration Product Line

This phase is where we build what we have called a *Migration Product Lines*, MPL. We define an MPL as a set of migrations that satisfy constraints specified by the domain expert user. The definition is similar to SPL, where each feature would be a migration in our case. This MPL allows us to obtain several valid migrations depending on the user's selection.

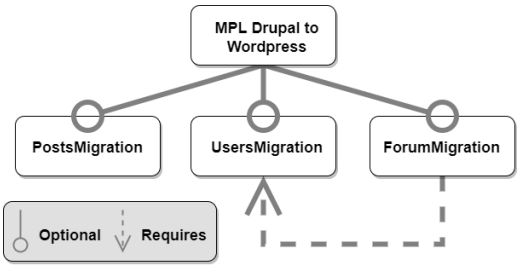

Figure 2: Example of Migration Product Line between Drupal-based system and WordPress-based system.

We propose the use of an MPL as part of our systematic process. We do not simply create the SQL *script* of a particular migration; each such migration, i.e., of a set of domain expert user-guided actions, is modeled in a migration product line. This MPL makes it possible to generate final databases based on the choice of features of that MPL. The final artifact is a SQL script composed of several intermediate scripts that are fully compatible with each other since it is the user himself who models the MPL and who configures the constraints between migrations. Each time the user generates a new migration, it is added to the MPL, specifying which restriction it has concerning the other migrations.

As a simple example, we start from the use case of migration between a Drupal-based system and a WordPress-based system in Figure 2. In our MPL there can exist the migrations *PostsMigration*, *UsersMigration* and *ForumMigration*. We assume that there is the particularity that *ForumMigration* cannot be performed if

UsersMigration is not applied first, but not the other way around. The *PostsMigration* migration is independent and has no restrictions of any kind. This would generate a total of 5 different final artifacts: (1) *PostsMigration*, (2) *PostsMigration - UsersMigration*, (3) *UsersMigration*, (4) *PostsMigration - UsersMigration - ForumMigration*, (5) *UsersMigration - ForumMigration*. And each of those final artifacts, i.e., *scripts* SQL, are valid configurations within our domain.

3 DEMO

Figure 3 shows the use of the console tool in manual selection mode. The tool displays the possible actions between the source and the target database in the first iteration. Depending on the selection of the action, it will show us a new set of available actions, being able to stop the execution at any time.

```
#####################################
original SDM file: sdm/A.xml
#####################################

Entity: Post (id = "post", number of attributes = 3)

        -- Attributes --
        Attribute: (varchar(25)) title
        Attribute: (text) body
        Attribute: (text) attribute_move

        -- Related entities --
        Relation: one(Post) to many(PostMeta)
        Entity: PostMeta (id = "post_meta", number of attributes = 1)

Entity: PostMeta (id = "post_meta", number of attributes = 1)

        -- Attributes --
        Attribute: (varchar(25)) author

        -- Related entities --
        Relation: one(Post) to many(PostMeta)
        Entity: Post (id = "post", number of attributes = 3)

-- All relations --
Relation: one(Post) to many(PostMeta)


#####################################
Available actions
#####################################

6 available actions

0 -> RenameAttributeAction
        rename attribute title to author in Post

1 -> RetypeAttributeAction
        retype attribute body to varchar(100) in Post

2 -> MoveAttributeAction
        move attribute author : varchar(25), from PostMeta to Post

3 -> CreateAttributeAction
        new attribute:  author : varchar(25) in entity Post

4 -> DeleteEntityAction
        delete entity:  post_meta

5 -> DeleteAttributeAction
        delete attribute body from Post


Select an available action ('q' for quit):
```

Figure 3: Example of the execution of the tool where we select actions from a set of available actions

4 CONCLUSIONS AND FUTURE WORK

The solution proposed in the previous section exemplifies, for a concrete case, that it is possible through a systematic analysis to obtain different valid migrations being the user an expert in the domain. This proposed tool avoids the cost of creating a script for each pair of versions between two databases, building a model that considers variability and generates new valid products. Our tool allows reasonably easy source or target database modifications without affecting the migration description process.

We propose an in-depth study of the following issues as future work:

- *Data variability study.* When migrating between different structures, how data is organized and treated can lead to integrity problems, data loss, inconsistencies, and incompatibilities. A systematic and structured analysis is proposed to establish thresholds and indicators to alert these potential problems.
- *Study of migrations between different paradigms.* We will study the possibility of implementing the solution to other non-relational databases such as MongoDB and analyze if there are inconsistencies during these transformations to avoid them or propose new solutions.
- *Generalizing migration product lines.* Currently the proposed solution allows the domain expert to generate a model representing the possible migrations of his problem. We will study whether it is possible to extend this horizon and cover different CMS and even different data storage paradigms in the same product line.
- *Data analysis.* Although the final artifact is a SQL script that allows automatic migration, intermediate artifacts of transformation models allow data to be exported to other formats such as CSV for more rigorous analysis through various technologies [2].
- *Inclusion in other systems.* We will analyze whether the tool can be included in other software management systems, such as continuous integration tools and backups.

ACKNOWLEDGMENTS

This work was supported by the Project (RTI2018-101204-B-C22, OPHELIA), funded by: FEDER/Ministry of Science and Innovation - State Research Agency); and the Junta de Andalucia COPERNICA (P20_01224) and METAMORFOSIS FEDER_US-1381375) projects.

REFERENCES

[1] Mustapha Bouakkaz and Yulia Strekalova. 2021. *Content Management System (CMS).* Springer. https://doi.org/10.1007/978-3-319-32001-4_43-1
[2] Koti and Yogi Reddy Maramreddy. 2020. A Study On Applications Of Data Mining. *International Journal of Scientific Technology Research* 9 (02 2020), 3385–3388.
[3] Annu Malik, Anju Sharma, and Mr. Vinod Saroha. 2013. Greedy Algorithm. *International Journal of Scientific and Research Publications* 3 (08 2013).
[4] Marek Novak. 2010. Easy Implementation of Domain Specific Language using XML. (01 2010).
[5] Marc Romanycia and Francis Pelletier. 1985. What is a heuristic? *Computational Intelligence* 1 (01 1985), 47 – 58. https://doi.org/10.1111/j.1467-8640.1985.tb00058.x
[6] Geetha S, Shasvat Shasvat, and Harsh Shah. 2021. Coded Websites vs Wordpress Websites. *International Journal of Advanced Research in Science, Communication and Technology* (12 2021), 212–216. https://doi.org/10.48175/IJARSCT-2140
[7] M. Sridharan, M. Mantyla, L. Rantala, and M. Claes. 2021. Data balancing improves self-admitted technical debt detection. In *Proceedings - 2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021.* 358–368.
[8] B. Terzić, S. Kordić, M. Čeliković, V. Dimitrieski, and I Luković. 2016. An Approach and DSL in support of Migration from relational to NoSQL Databases. *IICIST 2016 Proceedings* (2016), 179–184.