The TTC 2021 OCL2PSQL Case

Hoang Nguyen Phuoc Bao
ngpbhoang1406@gmail.com
Vietnamese-German University
Binh Duong, Vietnam

Antonio García-Domínguez
a.garcia-dominguez@aston.ac.uk
Aston University
Birmingham, United Kingdom

Manuel Clavel
manuel.clavel@vgu.edu.vn
Vietnamese-German University
Binh Duong, Vietnam

ABSTRACT

The Object Constraint Language (OCL) is a textual, declarative language used as part of the UML standard for specifying constraints and queries on models. As such, generating code from OCL expressions is part of an end-to-end model-driven development process. Certainly, this is the case for database-centric application development, where integrity constraints and queries can be naturally specified using OCL. Not surprisingly, there have been already several attempts to map OCL into SQL. In this case study, we invite participants to implement, using their own model-transformation methods, one of these mappings, called OCL2PSQL. We propose this case study as a showcase for different methods to prove their readiness for coping with moderately complex model transformations, by showing the usability, conciseness, and ease of understanding of their solutions when implementing a non-trivial subset of OCL2PSQL.

KEYWORDS

Model-transformation, OCL, SQL, TTC

ACM Reference Format:
Hoang Nguyen Phuoc Bao, Antonio García-Domínguez, and Manuel Clavel. 2021. The TTC 2021 OCL2PSQL Case. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Object Constraint Language (OCL) [7] is a textual language typically used, as part of the UML standard [8], for specifying constraints and queries on models. It is a pure specification language: expressions evaluate to values without changing anything in the underlying model. OCL is a strongly-typed language: expressions either have a primitive type, a class type, a tuple type, or a collection type. The language provides standard operators on primitive data, tuples, and collections. It also provides a dot-operator to access the properties of the objects, and several *iterators* to iterate over collections.

The Structured Query Language (SQL) [9] is a special-purpose programming language designed for managing data in relational database management systems (RDBMS). Its scope includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is, to a great extent, a declarative language, it also contains *stored-procedures*. These are routines stored in the database that may execute *loops* using the so-called *cursors*.

In the context of model-driven engineering, there exist several proposals for translating OCL into SQL [1, 3–5], which mostly differ in the way OCL iterators are translated. In particular, [3, 4] resort to imperative features of SQL (e.g. loops and cursors) for translating OCL iterators, while [1] introduces a mapping (OCL2PSQL) which only uses standard *subselects* and *joins* for translating OCL iterators[1].

Example 1.1. As an example of the transformations produced by OCL2PSQL, suppose that we want to know if, in a given scenario, *there is exactly one car*. We can formalize this query in OCL as follows:

Car.allInstances()−>size()=1

where we compare the number of objects in the class Car with an integer 1. OCL2PSQL translates this expression into an SQL select-statement as the comparison between the result of two-subqueries (e.g. `TEMP_left`.res and `TEMP_right`.res), representing respectively the result when evaluating each side of the comparison of the given OCL expression (e.g. Car.allInstances()−>size() and 1). Furthermore, the subquery `TEMP_left` returns the size of its subquery, aliased `TEMP_src`, which is the translation of the subexpression Car.allInstances().

```
SELECT TEMP_left.res = TEMP_right.res AS res, 1 AS val
FROM (
  SELECT COUNT(*) AS res, 1 AS val
  FROM (
    SELECT Car_id AS res, 1 AS val
    FROM Car
  ) AS TEMP_src
) AS TEMP_left
JOIN (
  SELECT 1 AS res, 1 AS val
) AS TEMP_right
```

The full recursive definition of OCL2PSQL can be found in [1], but we have included in Appendix A the subset of the OCL2PSQL definition of the expressions involved in this competition. The solution authors can also use Appendix A to understand the above transformation.

The *correctness* of the mapping is formulated as follows. Let e be an OCL expression (with no free variables) and let O be a scenario of its context model. Then, the evaluation of the expression e in the scenario O should return the same result that the execution of the query OCL2PSQL(e), i.e., the SQL query generated by

[1]The letter "P" in OCL2PSQL stands for *pure* SQL. The idea is that OCL2PSQL only uses the declarative features of SQL for mapping OCL expressions.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

OCL2PSQL from e, in the database OCL2PSQL(O), i.e., the database corresponding to O according to OCL2PSQL.[2]

The TTC 2021 OCL2PSQL Case welcomes participants to implement the subset of the OCL2PSQL mapping provided in Appendix A using their own model-transformation methods. This case study can serve as a showcase for different methods to prove their readiness to coping with moderately complex model-transformations, by showing the usability, conciseness, and understandability of their solutions when implementing the subset of the OCL2PSQL. Participants are also welcome to extend or modify the given subset of the OCL2PSQL mapping, or even to propose their own mapping from OCL to SQL, in which cases they should also provide convincing arguments that their solution is correct. More information about the main task will be provided in the later section. Finally, participants are most welcome to propose their own attributes of interests: for example, flexibility for multiple RDBMSs, or support for formal verification.

All resources for this case are available on Github [3]. Please follow the description in the footnote and create a pull request with your own solution after you have submitted your description to EasyChair.

The rest of the document is structured as follows: Section 2 describes the input and output of the OCL2PSQL transformation. Then, Section 3 provides the main task that should be tackled in a solution (participants are free to propose their own tasks of interest). Finally, Section 4 proposes the case evaluation scheme for the contest.

2 TRANSFORMATION DESCRIPTION

OCL2PSQL is a recently proposed mapping from OCL to SQL [1]. It addresses some of the challenges and limitations of previous OCL-to-SQL mappings, particularly with respect to the execution-time efficiency of the generated SQL queries. [2]

Next, we give a detail description of the input and output metamodels for the TTC 2021 OCL2PSQL Case. The input metamodels represent the part of OCL language that is covered in this competition. The output metamodel represents the part of the SQL language that is used by OCL2PSQL to translate the aforementioned part of OCL language. Obviously, for solutions that extend or modify the OCL2PSQL mapping, as well as for solutions that propose an entirely different mapping from OCL to SQL, the input and output metamodels presented here may need to be extended or modify accordingly.

2.1 Input Metamodel

OCL is a contextual language: its expressions are written in the context provided by a *data model*. Consequently, the input metamodel for OCL2PSQL can be seen as consisting of two, inter-related metamodels: namely, the metamodel for data models and the metamodel for OCL expressions.

[2]The OCL2PSQL mapping rests on an underlying mapping between data models and SQL database schema. The full definition of this mapping is also provided in [1] but it is not needed in this case.
[3]https://github.com/bluezio/ttc2021-ocl2psql (temporary, to be moved to main TTC Github organisation if accepted)

2.1.1 *Input metamodel for data models.*

For OCL2PSQL, a data model contains classes and associations. A class may have attributes and associations-ends. The multiplicity of an association-end is either 'one' or 'many'.

The data model metamodel for OCL2PSQL is shown in Figure 1. `DataModel` is the root element and contains a set of `Entitys`. Every `Entity` represents a class in the data model: it contains a set of `Attributes` and a set of `AssociationEnds`. Each `Attribute` represents an attribute of a class: it has a name and a type. Each `AssociationEnd` represents an association-end: it has a name, an association class name and an `Multiplicity` value. Each `AssociationEnd` is also linked to its opposite `AssociationEnd`, and with its target `Entity`.

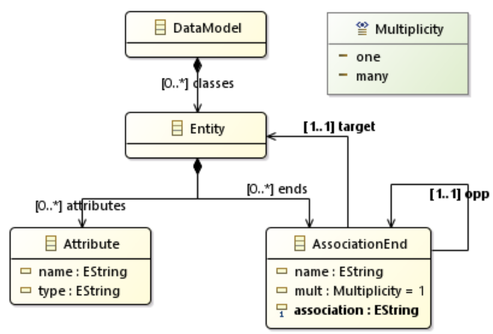

Figure 1: OCL2PSQL metamodel for data models.

2.1.2 *Input metamodel for OCL expressions.*

The definition of the OCL mapping as presented in Appendix A only covers a subset of the OCL language. For the OCL expressions involved in this competition, we have simplified the metamodel for OCL expressions to the minimum. For interested readers and solution authors who would like to extend or implement their own implementation, the class diagram of the OCL expression can be found in its specification document in [7]. The OCL2PSQL metamodel for OCL expressions in this competition is shown in Figure 2. In a nutshell, `OclExpression` is the root element. It is an abstract class. An `OclExpression` can be either a literal expression, a `CallExp`, a `VariableExp`, or a `TypeExp`. Next, we describe each of these classes.

A literal expression represents a literal value. In our case, it can be either an `IntegerLiteralExp`, a `StringLiteralExp`, or a `BooleanLiteralExp`. Each of these classes contains an attribute to represent, respectively, an integer, a string, or a boolean literal value.

A `TypeExp` represents a type expression. It contains an attribute `referredType` of type `Entity`, which belongs to the OCL2PSQL metamodel for data models.

A `VariableExp` represents a variable expression.

A `CallExp` represents an expression that consists of calling a feature over a `source`, which is represented by an `OclExpression`. `CallExp` is an abstract class: it can be either an `OperationCallExp`, a `PropertyCallExp`, an `AssociationClassCallExp`, or an `IteratorExp`.

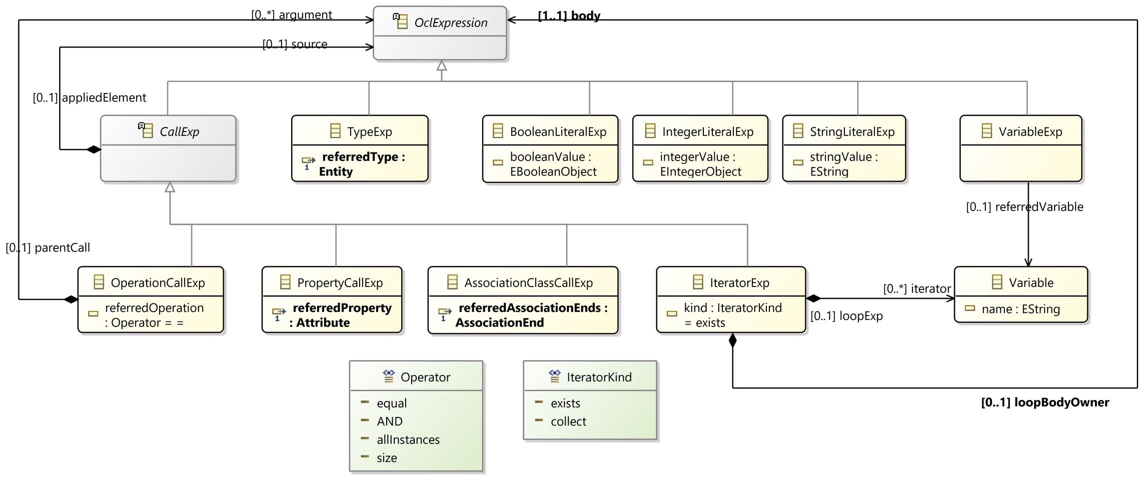

Figure 2: OCL2PSQL metamodel for OCL expression.

An `OperationCallExp` represents an expression that calls an operation over its `source`, possibly with arguments. For our competition, we only consider the equality comparison, i.e. =; conjunctive operation, i.e. AND; and two collections' operations, i.e. allInstances() and size().

A `PropertyCallExp` represents an expression that calls an attribute over a `source` object. The former is represented by an `Attribute` and the later is represented by an `Entity`; both belong to the OCL2PSQL metamodel for data models. OCL2PSQL only supports `PropertyCallExp` expressions whose `source` is an `VariableExp` expression. For example, given *c* is a `Variable` of type Car, *c*.color is a `PropertyCallExp` expression to get the color of the Car.

An `AssociationClassCallExp` represents an expression that calls an association-end over a `source` object. The former is represented by an `AssociationEnd` and the later is represented by an `Entity`; both belong to the OCL2PSQL metamodel for data models. OCL2PSQL only supports `AssociationClassCallExp` expressions whose `source` is an `VariableExp` expression. For example, given *c* is a `Variable` of type Car and owners is the association-end of Car, then *c*.owners is a `AssociationClassCallExp` expression to get the owners of the Car.

An `IteratorExp` represents an expression that calls an iterator over a `source` collection. The body of the iterator is represented by an `OclExpression` expression. The iterator-variable is represented by a `Variable`. In this competition, we support the following kinds of iterators: `exists`, and `collect`.

Figure 3 shows the object diagram of Car.allInstances()−>size()=1 in Example 1.1. It is an `OperationCallExp` with `referredOperation` as =, in which:

- The `source` is another expression of class `OperationCallExp` with `referredOperation` as `size`, representing the sub-expression Car.allInstances()−>size(). Furthermore, in the aforementioned sub-expression, the `source` is, yet, another expression of class `OperationCallExp` with `referredOperation` as `allInstances`, representing the sub-expression Car.allInstances(). Finally, in the aforementioned sub-expression, the `source` is a `TypeExp`, representing the sub-expression Car, which refers to the Car `Entity` in the data model.
- The `argument` is an `IntegerLiteralExp` with `integerValue` of 1.

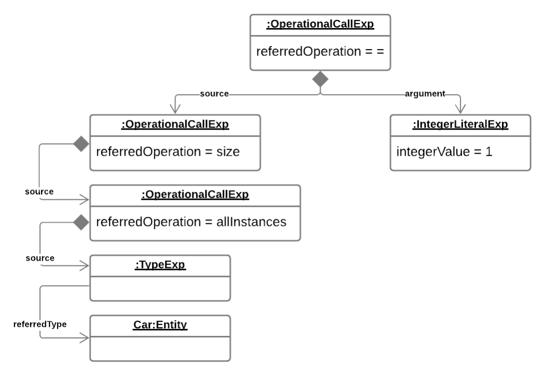

Figure 3: Object diagram of Car.allInstances()−>size()=1

2.2 Output Metamodel

For OCL2PSQL, an SQL query is a basic SQL `SELECT`-statement, which may contain subselects, `WHERE`-clauses, `GROUP BY`-clauses, and `JOIN`s. Figure 4 shows the overview diagram of a SQL `SELECT`-statement .

2.2.1 Output metamodel for SQL select-statements. The `SelectStatement` is the root element: it contains a `PlainSelect` , which represents the *body* of the select-statement.

A `PlainSelect` may contains the following objects: a list of `selItems` elements, each of type `SelectItem`; a `fromItem` element

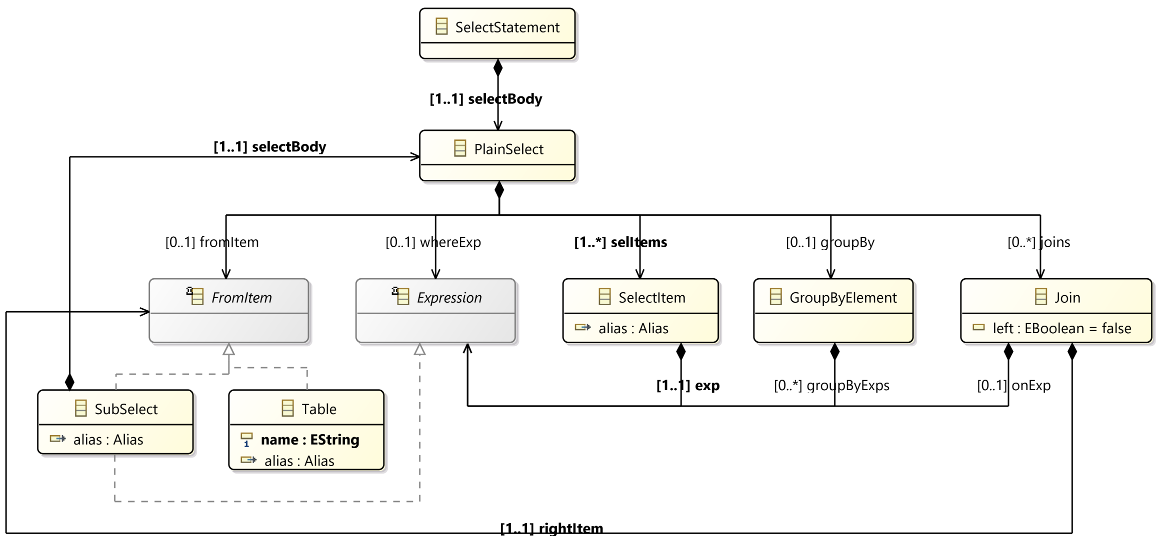

Figure 4: OCL2PSQL metamodel for SQL select-statements.

of type `FromItem`; a `whereExp` element of type `Expression`; a list of `joins` elements, each of type `Join`; and a `groupBy` element of type `GroupByElement`. Next, we describe each of these classes:

A `SelectItem` represents a *column* that the select-statement retrieves. It contains an `Expression` element and an `Alias` element.

A `FromItem` element represents the *table* or *subselect* from which the select-statement retrieves information. It is an interface. A `FromItem` element can be either a `Table` or a `SubSelect`. The former represents a *table*. The later represents a *subselect*.

A `whereExp` reference of type `Expression` represents a *where*-clause.

A `Join` element represents a *join* with a `rightItem` of type `FromItem`, possibly according to its element `onExp` of type `Expression`.

A `GroupByElement` element represents a *groupby*-clause. It contains `groupByExps`, a list of objects of type `Expression` that defines how the rows are to be *grouped by*.

`Expression` is an interface element which plays many role in a SQL-select statement. For the sake of simplicity, the realizations of `Expression` are hidden from Figure 4. Next, we describe these realizations which our cases will need.

A `LongValue` and `StringValue` represent, respectively, a literal integer and a literal string in SQL.

A `Column` represents a column of a table in SQL.

A `BinaryExpression` represents a binary expression in SQL. It contains a `leftExp` element and a `rightExp` element, both of type `Expression`. `BinaryExpression` is an abstract class. It can be either a logical expression, (`OrExpression` or `AndExpression`) , or a comparison expression (`EqualsToExpression` or `GreaterThanExpression`).

An `IsNullExpression` represents an `IS NULL`-expression in SQL. It contains an `Exp` element of type `Expression`.

A `CountAllFunction` represents a `COUNT(*)`-expression in SQL.

A `CaseExpression` represents a `CASE`-expression in SQL. It contains `whenClauses`, a list of objects of type `WhenClause`, representing `WHEN`-clauses in SQL.

A `SubSelect` represents a subselect-expression in SQL. It contains a `selectBody` of type `PlainSelect` .

For the sake of illustration, Figure 5 shows the object diagram of the following SQL-select statement:

```
SELECT COUNT(*) > 0 AS res
FROM Car AS c
WHERE c.color IS NULL
```

It is a `SelectStatement` with a `PlainSelect` as `selectBody`. The `PlainSelect` contains:

- A `SelectItem` element that represents the clause (SELECT) `COUNT(*) > 0 AS res`
  It contains a `GreaterThanExpression` expression, in which the `leftExp` is a `CountAllFunction` expression, and the `rightExp` is a `LongValue` expression with value 0. Furthermore, it has an `Alias` named res.
- A `Car` `Table` with an `Alias` named c, represents the clause (FROM) `Car AS c`.
- A `IsNullExpression` element that represents the clause (WHERE) `c.color IS NULL`
  It contains a `Column` color referred from the `Table` `Car` of the previous clause (notice that in this case, the alias c of the `Table` `Car` is used as a name for the table referred to the color column).

3 MAIN TASK

The main task for the participants in the TTC 2021 OCL2PSQL Case consists of implementing, using their own model-transformation methods, the subset of OCL2PSQL mapping as it is defined in Appendix A. Participants are free to extend or modify the OCL2PSQL

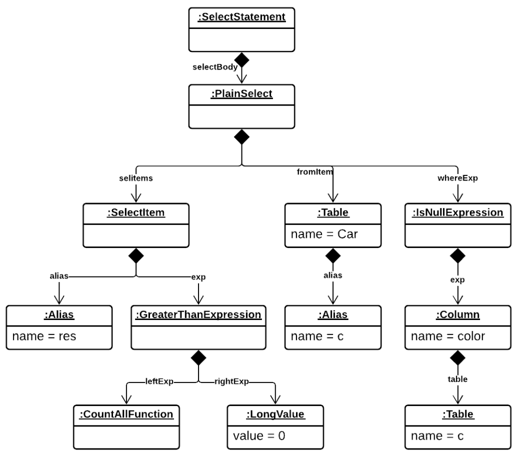

Figure 5: Object diagram of `SELECT COUNT(*) > 0 as res FROM Car c WHERE c.color IS NULL`

mapping, or even to propose their own mapping from OCL to SQL, in which case they should also provide convincing arguments that their solution is correct with respect to the semantics of OCL and SQL. [4].

During the contest, the participants will be presented with different *challenges* of increasing complexity. Each challenge will be an OCL2PSQL OCL expression, i.e., an instance of the OCL2PSQL metamodel for OCL. The *context* for all the challenges will be an OCL2PSQL data model, i.e., an instance of the OCL2PSQL metamodel for data models. Then, the participants will be asked to generate the *solutions* for these challenges, applying their own transformation rules. Very importantly: (i) each solution should be a valid SQL select-statement in the database schema corresponding to the given data model, according to the definition of the OCL2PSQL mapping; moreover, (ii) each solution should be an SQL select-statement returning a result-table with (at least) a column `res`. Then, when *executing* the solution for a challenge on a given scenario, this column `res` will be interpreted as holding the result of *evaluating* the given challenge in the same scenario. Finally, the solutions will be checked for *correctness*, using a set of selected *scenarios*.

For the participants' convenience, we have grouped the challenges into different stages. Each stage contains challenges that apply similar OCL2PSQL mapping rules, particularly:

- `Stage0` only requires the mapping rule for *literals*. The OCL expressions in this stage are context-free.
- `Stage1` is similar to `Stage1`, with additional mapping rules for `OperationalCallExp` (operator: equality and conjunction). The OCL expressions in this stage are also context-free.
- `Stage2` requires the mapping rule for `OperationalCallExp` (operator `allInstances`) and `TypeExp`. From this stage on, the OCL expressions are context-dependent, i.e., the underlying context model will be needed.
- `Stage3` is similar to `Stage2`, with additional mapping rule for `OperationalCallExp` (operator: `size` and equality).
- `Stage4` is similar to `Stage3`, with additional mapping rule for `VariableExp` and `IteratorExp` (kind: `collect`).
- `Stage5` is similar to `Stage4`, with additional mapping rule for `PropertyCallExp`.
- `Stage6` is similar to `Stage4`, with additional mapping rule for `AssociationClassCallExp`.
- `Stage7` is similar to `Stage5` and `Stage6`, with additional mapping rule for `IteratorExp` (kind: `exists`).
- `Stage8` is a more complex version of `Stage7`, with nested `IteratorExp` of kind `exists`.

For the purpose of testing, the participants can find the following material in the case materials repository:

- In the `docs` folder, the file `challenges.txt` contains a list of *challenges* grouped in the aforementioned *stages*. Each stage has a unique number, and each challenge within a stage has also a unique number. The greater the number of a stage, the greater its complexity. The *context* for all challenges in `challenges.txt` is the data model `CarPerson` shown in Figure 6.
  In the same folder, the file `scenarios.txt` contains a list of *scenarios*. Each scenario describes an instance of the data model `CarPerson`. Then, for each scenario, and each (relevant) `stage/challenge` listed in `challenges.txt`, the file `scenarios.txt` contains the *correct* result: i.e., the expected SQL result that corresponds to the evaluation of the given *stage*/`challenge` in the given scenario.
- The folder `input` contains the challenges listed in `challenges.txt` in XMI format. More specifically, each file `Stage`*i*`Challenge`*j*`.xmi` contains the representation, in XMI-format, of the challenge *j* within the stage *i* in the file `challenges.txt`.
  In the same folder, the file `CarPerson.xmi` contains the data model `CarPerson` in XMI-format.
- In the folder `metamodels`, the file `ocl.ecore` contains the EMF implementation of the OCL2PSQL metamodel for OCL expressions. Also in the same folder, the file `sql.ecore` contains the EMF implementation of the OCL2PSQL metamodel for SQL-select statements.

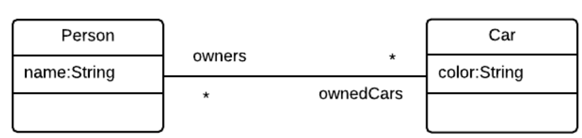

Figure 6: The `CarPerson` data model.

4 BENCHMARK FRAMEWORK

The case resources on Github include an automated benchmark framework for systematic measurement of the performance and correctness of the various solutions. It is based on the framework

[4]For the participants who would like to extend their implementation beyond the subset of OCL language provided for our competition, please revise the full version of our OCL2PSQL mapping in [1] with the "fixes" included in Appendix B.

Listing 1: `solution.ini` file for the `ReferenceXMI` solution

```
[build]
default=mvn compile
skipTests=mvn compile

[run]
cmd=mvn −f pom.xml −quiet −Pxmi exec:exec
```

of the TTC 2017 Smart Grid case [6], without the visualisation components. Solution authors are heavily recommended to adapt their solutions to this framework, to allow for the easier integration and comparison of the various solutions.

The configuration of the benchmark framework for the TTC 2021 OCL2PSQL case is stored in the file `config.json` inside the folder `config`. This file includes the definitions of the various stages and challenges, the name of the tools to be run, the number of repetitions to be applied, the timeout in milliseconds for each execution and the connection information for the local MySQL database. Currently, the file `config.json` has already contained the stages and challenges listed in the file `challenges.txt`.

In the folder `docker`, the `Dockerfile` contains the instruction to build a MySQL 5.7 Docker image that contains all the SQL data scenarios of `CarPerson` database corresponding to the ones listed in `scenarios.txt`. This image is currently used for building databases to test the correctness of the reference solution. Solution authors can use either the image we provide or their own local MySQL database installation, in which they would need to change the information in the `config`.

4.1 Solution requirements

All solutions must be forks of the main Github project, and should be submitted as pull requests after the descriptions have been uploaded to EasyChair.

All solutions should be in a subdirectory of the `solutions` folder, and inside this subdirectory they should include a `solution.ini` file describing how the solution should be built and how it should be run. As an example, Listing 1 shows the file for the reference solution. The build section provides the `default` and `skipTests` fields for specifying how to build and test, and how to simply build, respectively. In the `run` section, the `cmd` field specifies the command to run the solution.

Solutions should print to their standard output streams a sequence of lines with the following fields, separated by semicolons:

- **Tool**: name of the tool.
- **Stage**: integer with the stage within the case whose challenge is being solved.
- **Challenge**: integer with the challenge within the stage which is being solved.
- **RunIndex**: integer with the current repetition of the transformation.
- **MetricName**: may be "TransformTimeNanos", "TestTimeNanos", or "Scenario*ID*" where *ID* is the identifier of the scenario under test.
- **MetricValue**: the value of the metric:
  – For "TransformTimeNanos", an integer with the nanoseconds spent performing the transformation.
  – For "TestTimeNanos", an integer with the nanoseconds spent testing the correctness of the transformation through executing the transformed SQL-select statement on different database scenarios.
  – For metrics following the "ScenarioID" pattern, a string of either "passed" or "failed", indicating either the correctness of the transformation on that scenario.

The repetition of the transformation is handled by the framework. Moreover, for every repetition, the framework provides, through environment variables, the following information: the run index, stage number and challenge number, the OCL expression corresponding to the challenge in plaintext, as well as the file path of that expression in XMI-format, and the file path of the context of the challenge, also in XMI-format. More specifically, the available environment variables are:

- **MySQLUsername**: the username of the local MySQL database system on which the statement will be run.
- **MySQLPassword**: the password of the given user.
- **MySQLPort**: the port number the local MySQL database system.
- **StageIndex**: the index of the stage whose challenge is to be run.
- **ChallengeIndex**: the index of the challenge within the stage which will be run.
- **OCLQuery**: the OCL expression, in text-format, corresponding to the challenge to be run.
- **PathToOCLXMI**: the absolute path to the file containing the OCL expression, in XMI-format, corresponding to the challenge to be run.
- **PathToSchemaXMI**: the absolute path to the file containing the SQL schema, in XMI-format, corresponding to the context (data model) of the challenges to be run.
- **RunIndex**: the index of the repetition to be run.
- **Tool**: the name of the tool (the name of the `solutions` subfolder).

Solution authors may wish to consult the reference solution for guidance on how to use the various environment variables and how to test the correctness of your transformations. Solution authors are free to reuse the source code of this reference solution for these aspects (e.g. the `CaseLauncher` and `Configuration` classes), as well as the `lib/sql.jar` library, in the reference solution , that parses the SQL-select statement from XMI model to plaintext. The reference solution uses Maven to retrieve the appropriate libraries for communicating with the our own implementation of OCL2PSQL. In addition, we have also installed locally additional libraries in folder `lib` using a shell script. The instruction for running the reference solution can be found on the benchmark repository.

4.2 Running the benchmark

The benchmark framework needs Python 3.3 or later to be installed, and the reference solution requires Maven 3 and Java 8 or later. Solution authors are free to use alternative frameworks and programming languages, as long as these dependencies are explicitly documented. For the final evaluation, it is planned to construct a

Docker image with all solutions, and this will require installing those dependencies into the image.

If all dependencies are installed, the benchmark can be run with `python scripts/run.py` (potentially `python3` if Python 2.x is installed globally in the same system).

5 EVALUATION

The benchmark framework will provide independent measurements of the correctness, completeness, and time usage of the solutions provided by the participants. Attendees to the contest will evaluate the usability, conciseness, and understandability of the transformation rules that define the different solutions, as well as the other attributes of interest that the solution providers may want to focus in. In this regard, although some solutions may not be entirely complete or may be hard to understand, they may still serve as examples of active research areas within model transformations that the community may wish to showcase. To recognize these contributions, an audience-driven "Most Promising" award will be given.

REFERENCES

[1] H. Nguyen Phuoc Bao and M. Clavel. 2019. OCL2PSQL: An OCL-to-SQL Code-Generator for Model-Driven Engineering. In *Future Data and Security Engineering - 6th International Conference, FDSE 2019, Proceedings (Lecture Notes in Computer Science)*, T. Khanh Dang, J. Küng, M. Takizawa, and S. Ha Bui (Eds.), Vol. 11814. Springer, 185–203.

[2] M. Clavel and H. Nguyen Phuoc Bao. 2019. Mapping OCL into SQL: Challenges and Opportunities Ahead. In *19th International Workshop in OCL and Textual Modeling (OCL 2019) co-located with MODELS 2019 (CEUR Workshop Proceedings)*, A. D. Brucker, G. Daniel, and F. Jouault (Eds.), Vol. 2513. CEUR-WS.org, 3–16.

[3] M. Egea and C. Dania. 2019. SQL-PL4OCL: an automatic code generator from OCL to SQL procedural language. *Software and Systems Modeling* 18, 1 (2019), 769–791.

[4] M. Egea, C. Dania, and M. Clavel. 2010. MySQL4OCL: A Stored Procedure-Based MySQL Code Generator for OCL. *ECEASST* 36 (2010).

[5] F. Heidenreich, C. Wende, and B. Demuth. 2008. A Framework for Generating Query Language Code from OCL Invariants. *ECEASST* 9 (2008).

[6] Georg Hinkel. 2017. An NMF solution to the Smart Grid Case at the TTC 2017. In *Proceedings of the 10th Transformation Tool Contest (TTC 2017), co-located with the 2017 Software Technologies: Applications and Foundations (STAF 2017), Marburg, Germany, July 21, 2017 (CEUR Workshop Proceedings)*, Antonio García-Domínguez, Georg Hinkel, and Filip Krikava (Eds.), Vol. 2026. CEUR-WS.org, 13–17. http://ceur-ws.org/Vol-2026/paper5.pdf

[7] Object Management Group. 2014. *Object Constraint Language specification Version 2.4.* Technical Report.

[8] Object Management Group. 2017. *Unified Modeling Language.* Technical Report.

[9] International Organization for Standardization. 2011. *ISO/IEC 9075-(1–10) Information technology – Database languages – SQL.* Technical Report.

A THE MAPPING OCL2PSQL IN A NUTSHELL

The mapping OCL2PSQL is defined recursively over the structure of OCL expressions. To describe the key idea underlying its definition, and to illustrate it with the presentation of some recursive cases, we need to introduce first some notation.

Notation. Let qry be an SQL query. Let db be an SQL database. Then, we denote by $\mathrm{Exec}(qry, db)$ the result of executing qry on db. Let e be an OCL expression. Then, we denote by $\mathrm{FVars}(e)$ the set of variables that occur *free* in e, i.e., that are not *bound* by any iterator. Let e be an OCL expression, and let v be a variable introduced in e by an iterator expression s $->$*iter* $(v \mid b\ )$. Then, we denote by $\mathrm{src}_e(v)$ the *source* s of v in e. Let e be an OCL expression and let e' be a subexpression of e. Then, we denote by $\mathrm{SVars}_e(e')$ the set of variables which (the value of) e' depends on, and is defined as follows:

$$\mathrm{SVars}_e(e') = \bigcup_{v \in \mathrm{FVars}(e')} \{v\} \cup \mathrm{SVars}_e(\mathrm{src}_e(v)).$$

Let e be an OCL expression, such that $\mathrm{FVars}(e) = \emptyset$. Let $\mathcal{O}$ be a scenario. Then, we denote by $\mathrm{Eval}(e, \mathcal{O})$ the result of *evaluating* e in $\mathcal{O}$.

Finally, let $\mathcal{D}$ be a data model. Then, we denote by $\mathrm{map}(\mathcal{D})$ the SQL database schemata corresponding to $\mathcal{D}$, according to OCL2PSQL. Let $\mathcal{D}$ be a data model, and let $\mathcal{O}$ be a scenario of $\mathcal{D}$. Then, we denote by $\mathrm{map}(\mathcal{O})$ the instance of $\mathcal{D}$ corresponding to $\mathcal{O}$, according to OCL2PSQL. Let e be an OCL expression, let e' be a subexpression of e. Then, we denote by $\mathrm{map}_e(e')$ the SQL query corresponding to e', according to OCL2PSQL.

Definition: key idea and some cases. The different recursive cases follow the same design principle: namely, let e be an OCL2PSQL-expression, let e' be a subexpression of e, and let $\mathcal{O}$ be a scenario. Then, $\mathrm{Exec}(\mathrm{map}_e(e'), \mathrm{map}(\mathcal{O}))$ returns a table, with a column `res`, a column `val`, and, for each $v \in \mathrm{SVars}_e(e')$, a column `ref_`$v$. Informally, for each row in this table: (i) the columns `ref_`v contain a valid "instantiation" for the iterator variables of which the evaluation of e' depends on (if any); (ii) the column `val` contains 0 when evaluating the expression e', with the "instantiation" represented by the columns `ref_`v, evaluates to the *empty set*; otherwise, the column `val` contains 1; (iii) when the column `val` contains 1, the column `res` contains the result of evaluating the expression e' with the "instantiation" represented by the columns `ref_`v; when the column `val` contains 0, the value contained in the column `res` is not meaningful.

We define the recursive definition of OCL2PSQL mappings that will be used in our competition. The definition here was taken from the original paper and has already included the corrigenda in Appendix B.

Literal strings (correspondingly: integers, booleans)

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = l$, where l is either a literal string. Then,

$\mathrm{map}_e(l)$ =
`SELECT` l `as res, 1 as val`

Variables

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = v$, where v is a variable. Then,

$\mathrm{map}_e(v)$ =
`SELECT`
  `TEMP_dmn.res as res,`
  `TEMP_dmn.res as ref_`v`,`
  `TEMP_dmn.val as val,`
  `TEMP_dmn.ref_`v' `as ref_`v', for each $v' \in \mathrm{SVars}_e(\mathrm{src}(v))$
`FROM (`$\mathrm{map}_e(\mathrm{src}(v))$`) as TEMP_dmn`

Attribute-expressions

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = v$ $.att$, where v is a variable of class-type c and att is an attribute of the class c. Then,

```
map_e(v .att) =
SELECT
  c.att as res,
  TEMP_obj.val as val,
  TEMP_obj.ref_v' as ref_v', for each v' ∈ SVars_e(v)
FROM (map_e(v)) as TEMP_obj
LEFT JOIN c
ON TEMP_obj.ref_v = c.c_id AND TEMP_obj.val = 1
```

Association-ends-expressions

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = v$ $.ase$, where v is a variable of class-type c, and ase is an association-end of the class c.

Let Assoc(ase) be the association to which ase belongs, and let Oppos(ase) be the association-end at the opposite end of ase in Assoc(ase). Then,

```
map_e(v .ase) =
SELECT
  Assoc(ase).ase as res,
  CASE Assoc(ase).Oppos(ase) IS NULL
    WHEN 1 THEN 0
    ELSE 1 END as val,
  TEMP_src.ref_v' as ref_v', for each v' ∈ SVars_e(v)
FROM (map_e(v)) as TEMP_src
LEFT JOIN Assoc(ase)
ON TEMP_src.ref_v = Assoc(ase).Oppos(ase)
```

AllInstances-expressions

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = c$ .allInstances(), where c is a class type. Then,

```
map_e(c .allInstances())=
SELECT c_id as res, 1 as val FROM c
```

size-expressions

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = s$ −>size(). We need to consider the following cases:

- $\mathrm{FVars}(e') = \emptyset$. Then,

```
map_e(s −>size()) =
SELECT
  COUNT(*) as res,
  1 as val
FROM (map_e(s)) AS TEMP_src.
```

- $\mathrm{FVars}(e') \neq \emptyset$, Then,

```
map_e(s −>size()) =
SELECT
  CASE TEMP_src.val = 0
    WHEN 1 THEN 0
    ELSE COUNT(*) END as res,
  TEMP_src.ref_v as ref_v, for each v ∈ SVars_e(s)
  1 as val
FROM (map_e(s)) AS TEMP_src
GROUP BY
  TEMP_src.ref_v, for each v ∈ SVars_e(s),
  TEMP_src.val.
```

=-expressions (correspondingly, and-expressions)

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = (l = r)$. For our competition, we only need to consider the following cases:

- $\mathrm{FVars}(l) = \mathrm{FVars}(r) = \emptyset$. Then,

```
map_e(l =r) =
SELECT
  TEMP_left.res = TEMP_right.res as res,
  1 as val
FROM
  (map_e(l)) AS TEMP_left,
  (map_e(r)) AS TEMP_right
```

- $\mathrm{FVars}(l) \neq \emptyset$, $\mathrm{SVars}(r) \subseteq \mathrm{SVars}(l)$. Then,

```
map_e(l =r) =
SELECT
  TEMP_left.res = TEMP_right.res as res,
  CASE
    TEMP_left.val = 0 OR TEMP_right.val = 0
    WHEN 1 THEN 0
    ELSE 1 END as val,
  TEMP_left.ref_v as ref_v, for each v ∈ SVars_e(l)
FROM (map_e(l)) AS TEMP_left
[LEFT] JOIN (map_e(r)) AS TEMP_right
[ON TEMP_left.ref_v = TEMP_right.ref_v,
    for each v ∈ SVars_e(l) ∩ SVars_e(r)].
```

collect-expressions

Let e be an OCL expression. Let e' be a subexpression of e. Let $e' = s$ −>collect($v \mid b$). For our competition, we only need to consider the following case:

- $v \in \mathrm{FVars}(b)$ and $\mathrm{FVars}(e') = \emptyset$.

```
SELECT TEMP_body.res as res,
  TEMP_body.val as val,
FROM (map_e(b)) as TEMP_body
```

exists-expressions

Let e be an OCL2PSQL-expression. Let e' be a subexpression of e. Let $e' = s$ −>exists($v \mid b$). For our competition, we only need to consider the following cases:

- $v \in \mathrm{FVars}(b)$ and $\mathrm{FVars}(e') = \emptyset$. Then

```
SELECT
  COUNT(*) > 0 as res,
  1 as val
FROM (map_e(b)) as TEMP_body
WHERE TEMP_body.res = 1
```

- $v \in \mathrm{FVars}(b)$ and $\mathrm{FVars}(e') \neq \emptyset$. Then

```
SELECT
  CASE TEMP_body.ref_v IS NULL
    WHEN 1 THEN 0
    ELSE TEMP_body.res END as res,
```

`1 as val,`
`TEMP_src.ref_`v' `as ref_`v',
for each $v' \in \mathrm{SVars}(s)$,
`TEMP_body.ref_`v' `as ref_`v',
for each $v' \in \mathrm{SVars}(b) \setminus \mathrm{SVars}(s) \setminus \{v\}$
`FROM (`$\mathrm{map}_e(s)$`) as TEMP_src`
`LEFT JOIN (`
`SELECT COUNT(*) > 0 as res,`
`TEMP_body.ref_`v' `as ref_`v',
for each $v' \in \mathrm{SVars}(b)$
`FROM (`$\mathrm{map}_e(b)$`) as TEMP_body`
`WHERE TEMP_body.res = 1`
`GROUP BY TEMP_body.ref_`v',
for each $v' \in \mathrm{SVars}(b) \setminus \{v\}$
`) as TEMP_body`
`ON TEMP_src.ref_`v' = `TEMP_body.ref_`v',
for each $v' \in \mathrm{SVars}(s)$

B CORRIGENDUM

In [1, Section 4.3], in the second case considered in the definition of the mapping for `Exists`-expressions instead of:

- $v \in \mathrm{FVars}(b)$ and $\mathrm{FVars}(e') \neq \emptyset$. Then

`SELECT`
`CASE` $\underline{\texttt{TEMP\_src.ref\_}v}$ `IS NULL`
`WHEN 1 THEN 0`
`ELSE` $\underline{\texttt{TEMP.res}}$ `END as res,`
...
`LEFT JOIN (`
`SELECT COUNT(*) > 0 as res,`
`TEMP_body.ref_`v' `as ref_`v',
for each $v' \in \underline{\mathrm{SVars}(b) \setminus \{v\}}$

it should read:

- $v \in \mathrm{FVars}(b)$ and $\mathrm{FVars}(e') \neq \emptyset$. Then

`SELECT`
`CASE` $\underline{\texttt{TEMP\_body.ref\_}v}$ `IS NULL`
`WHEN 1 THEN 0`
`ELSE` $\underline{\texttt{TEMP\_body.res}}$ `END as res,`
...
`LEFT JOIN (`
`SELECT COUNT(*) > 0 as res,`
`TEMP_body.ref_`v' `as ref_`v',
for each $v' \in \underline{\mathrm{SVars}(b)}$

And similar errors should be corrected in [1, Section 4.3], in the second case considered in the definition of the mapping for `ForAll`-expressions.