TYPHON
POLYGLOT AND HYBRID PERSISTENCE ARCHITECTURES FOR BIG DATA ANALYTICS

Project Number 780251

D2.5 TyphonML Model Analysis and Reasoning Tools

Version 1.0
23 December 2019
Final

Public Distribution

University of L'Aquila

Project Partners: **Alpha Bank**, **ATB**, **Centrum Wiskunde & Informatica**, **CLMS**, **Edge Hill University**, **GMV**, **OTE**, **SWAT.Engineering**, **The Open Group**, **University of L'Aquila**, **University of Namur**, **University of York**, **Volkswagen**

Project Partner Contact Information

Alpha Bank Vasilis Kapordelis 40 Stadiou Street 102 52 Athens Greece Tel: +30 210 517 5974 E-mail: vasileios.kapordelis@alpha.gr	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany Tel: +49 421 22092 0 E-mail: scholze@atb-bremen.de
Centrum Wiskunde & Informatica Tijs van der Storm Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 20 592 9333 E-mail: storm@cwi.nl	**CLMS** Antonis Mygiakis Mavrommataion 39 104 34 Athens Greece Tel: +30 210 619 9058 E-mail: a.mygiakis@clmsuk.com
Edge Hill University Yannis Korkontzelos St Helens Road Ormskirk L39 4QP United Kingdom Tel: +44 1695 654393 E-mail: yannis.korkontzelos@edgehill.ac.uk	**GMV Aerospace and Defence** Almudena Sánchez González Calle Isaac Newton 11 28760 Tres Cantos Spain Tel: +34 91 807 2100 E-mail: asanchez@gmv.com
OTE Theodoros E. Mavroeidakos 99 Kifissias Avenue 151 24 Athens Greece Tel: +30 697 814 7618 E-mail: tmavroeid@ote.gr	**SWAT.Engineering** Davy Landman Science Park 123 1098 XG Amsterdam Netherlands Tel: +31 633754110 E-mail: davy.landman@swat.engineering
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org	**University of L′Aquila** Davide Di Ruscio Piazza Vincenzo Rivera 1 67100 L'Aquila Italy Tel: +39 0862 433735 E-mail: davide.diruscio@univaq.it
University of Namur Anthony Cleve Rue de Bruxelles 61 5000 Namur Belgium Tel: +32 8 172 4963 E-mail: anthony.cleve@unamur.be	**University of York** Dimitris Kolovos Deramore Lane York YO10 5GH United Kingdom Tel: +44 1904 325167 E-mail: dimitris.kolovos@york.ac.uk
Volkswagen Behrang Monajemi Berliner Ring 2 38440 Wolfsburg Germany Tel: +49 5361 9-994313 E-mail: behrang.monajemi@volkswagen.de	

Document Control

Version	Status	Date
0.1	Document outline	28 November 2019
0.2	First draft	7 December 2019
0.7	First full draft	16 December 2019
0.8	Further editing draft	18 December 2019
1.0	Final updates after partner reviews	23 December 2019

Table of Contents

Executive Summary

In enterprise applications the use of databases can affect some important quality parameters such as performance and maintainability. A smell in a database system indicates the violation of the recommended best practices and potentially affect in negative way the quality of the considered software system. For this reason, the early detection of smells in developing database schema could lead to high-quality software systems and even enhance some quality performances. This document presents tools that have been developed in the context of WP2 to support the specification and detection of TyphonML smells.

1 Introduction

Work package 2 is focusing on the development of the novel TyphonML modeling language and supporting tools for the specification of polystores and their early analysis before their actual deployment. TyphonML permits to abstract over the specificities of the underlying technologies and to specify the conceptual entities, which are subsequently mapped on the different kinds of databases available in the polystore being considered. Further than defining models that are syntactically correct, the TyphonML tools permit to reason about the properties of TyphonML design models in preparation of their transformation to deployment models expressed in the Typhon Deployment Language (TyphonDL) developed in WP3. To this end, in this deliverable we present results of WP2 related to the following task (from the TYPHON DoW):

> *Task 2.4: Analysis and Reasoning on TyphonML Models.* This task will produce algorithms for analysing and reasoning about the feasibility and properties of polystores modelled using TyphonML. The aim of this task is to provide engineers with preliminary feedback before assembling and deploying modelled polystores, and to pave the way for the automated transformation to TyphonDL which will be undertaken in WP3.

This task is extremely connected with the following WP3 task:

> *Task 3.3: Transformation of TyphonML Design Models to Deployment (TyphonDL) Models* This task will produce automated transformations for mapping polystore design (TyphonML) models to deployment (TyphonDL) models, which will specify the way to assemble virtual machines (or instances) that contain all the software components needed to support a polystore (e.g. operating system, system services, configured persistence back-ends). The transformations between TyphonML and the TyphonDL models will be developed using contemporary model transformation languages like ATL and ETL.

Both *Task 2.4* and *Task 3.3* (both under the main responsibility of UDA and ending at M24) are related to the final goal of reasoning about the feasibility and properties of TyphonML models, and to eventually enable the deployment of the modeled polystores with respect to functional and non-functional requirements defined by the system developer. To this end, the concepts and tools that are presented in this deliverable can be seen as a building block of a more general process defined in the context of Task 3.3 and presented in the deliverable D3.3 [4].

In this deliverable we focus on the notion of TyphonML smells that can potentially affect in negative way the quality of the modeled system e.g., with respect to quality parameters like maintainability. A set of smells has been defined and corresponding detection tools have been also presented. The approach has been designed in an extensible way so that whenever new smells need to be recognized in source TyphonML models, the developer can easily extend the detection capabilities of the smell detection tools.

1.1 Structure of the deliverable

The document is organized as follows:

- Section 2 sets the context of the presented work by locating the *Smell detection* phase in the analysis and reasoning process defined in the deliverable D3.3 [4]

- Section 3 presents a catalogue of TyphonML smells. Moreover, the corresponding smell detectors are presented.
- Section 4 concludes the document and provides the reader with an overview of the next steps.

2 Analysis of TyphonML models

TyphonML language and supporting tools permit modelers to specify both the conceptual entities of the application being developed and how they have to be mapped on the available DB infrastructures. To mitigate the difficulties of properly mapping conceptual data entities to the most appropriate database systems, in the deliverable D3.3 [4] an enhancement of the TyphonML language and supporting tools has been presented. In particular, as shown in Fig. 1, modelers have now the possibility to annotate conceptual entities with functional and non-functional requirements (see TyphonML$_{CE}$ in Fig. 1). Then, *feasibility checks* are applied to check if the given requirements can be met with respect to the available database technologies. In the *mapping generation* phase, the found solution is actualized by generating the TyphonML$_{DBMap}$ part of the TyphonML being specified. The corresponding TyphonDL specificaiton is automatically obtained by means of the *TyphonDL generation* step.

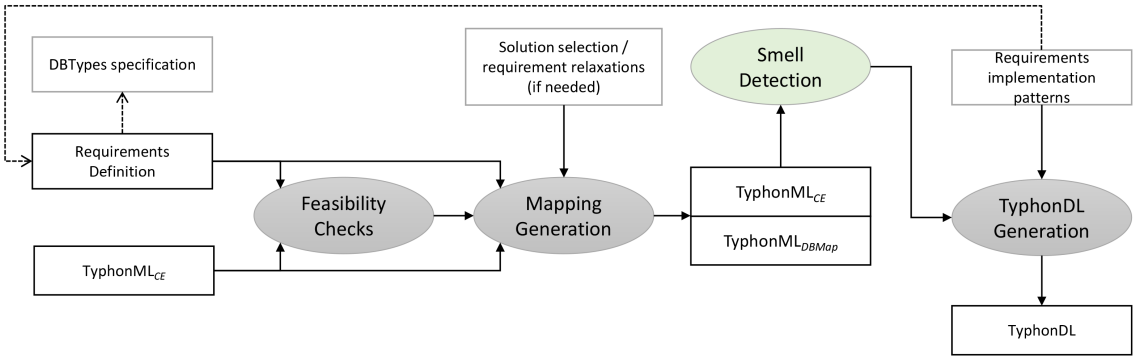

Figure 1: Proposed analysis and reasoning process (extension of that in D3.3 [4])

As discussed in [5], similarly to code, database schemas are also prone to smells (i.e., best practice violations) that can introduce quality problems in a software system. An example of smell that can occur in TyphonML is when one of the specified entity references is a containment and it is also mapped to a relational database. This is the case of the reference `review` of the entity `Product` as specified in line 9 of Listing 1. According to the given mapping, the `Product` entity is mapped to a relational database (see lines 49–54 in Listing 1). Although this is not a real error, the best choice would be to avoid mapping this containment to a relational database, and use document databases instead.

```
1  -- Conceptual entities
2  entity Review{
3          product -> Product[1]
4  }
5
6  entity Product{
7          name : String
8          description : String
9          review :-> Review."Review.product"[0..*]
10         orders -> Order[0..*]
11         photo : Jpeg;
12 }
13
14 entity Order{
15         date : Date
16         totalAmount : Int
```

```
17          products -> Product."Product.orders"[0..*]
18          users -> User."User.orders"[1]
19          paidWith -> CreditCard[1]
20  }
21
22  entity User{
23          name : String
24          surname : String
25          paymentsDetails :-> CreditCard[0..*]
26          orders -> Order[0..*]
27  }
28
29  entity CreditCard{
30          number : String
31          expiryDate : Date
32  }
33
34  -- Specification of data mappings
35  relationaldb RelationalDatabase{
36          tables{
37                  table {
38                          OrderDB : Order
39                          index orderIndex {
40                                  attributes ("Order.date")
41                          }
42                  }
43                  table {
44                          UserDB : User
45                          index  userIndex{
46                                          attributes ("User.name")
47                          }
48                  }
49                  table {
50                          ProductDB : Product
51                          index productIndex{
52                                  attributes ("Product.name")
53                          }
54                  }
55                  table {
56                          CreditCardDB : CreditCard
57                          index creditCardIndex{
58                                  attributes ("CreditCard.number")
59                          }
60                          }
61                  }
62          }
63  }
64
65  graphdb ConcordanceDB  {
66          nodes {
67                  node ProductNode!Product {
68                          name = "Product.name"
69                  }
```

```
70          }
71          edges {
72                  edge concordance {
73                          from ProductNode
74                          to ProductNode
75                          labels {
76                                  weight:int
77                          }
78                  }
79          }
80  }
```

Listing 1: Sample TyphonML specification

In the deliverable D2.4 [3] we started the investigation about how to support the automated detection of smells in TyphonML models. Since then, we explored further and we conceived the *Smell Detection* phase, which is shown in Fig. 1, and it is an enhancement of the process presented in D3.3 (which focuses more on feasability checks, mapping generation, and on the corresponding automated TyphonDL generation).

3 Definition and detection of TyphonML smells

In enterprise applications the use of databases can affect some important quality parameters such as performance and maintainability. A smell in a database system indicates the violation of the recommended best practices and potentially affect in negative way the quality of the considered software system. For this reason, the early detection of smells in developing database schema could lead to high-quality software systems and even enhance some quality performances [5]. According to the work presented in [5] datatabase smells can be categorized in three categories:

- *Schema smells*, which arise due to poor schema design;
- *Query smells*, which arise from poorly written SQL queries e.g., when a query references at least one non-grouped column in the presence of group by clause;
- *Data smells*, which arise from poor data handling in databases (for instance, when 'O' is used instead of '0' in 7O34)

3.1 TyphonML smells

Since TyphonML is supposed to be used for defining the structure of the data to be managed by the polystore being developed, in the following we focus on schema smells. According to the performed investigations, the following smells might affect the quality of TyphonML specifications:

SM1 *Containment relations in relational databases*: for performance purposes containment relations should be managed by means of NoSQL databases e.g., document databases;

SM2 *God entity*: this smell arises when an entity contains an excessive number of attributes. The excessive number of attributes tends to violate the principles of normalization which in turn introduces a variety of problems. Additionally, it impacts the maintainability of the database;

SM3 *Overloaded attribute name*: this smell occurs when two or more attributes are defined with identical names but as distinct data types in different entities. Identical names with different data types create confusion and could lead to subtle bugs in queries;

SM4 *Meaningless name*: this smell occurs when an entity or an attribute name is cryptic or meaningless. Best practice for naming is to use well defined and consistent names for entities, attributes, and references.

SM5 *Use of plural for table names*, it is recommended to use singular e.g., use `StudentCourse` instead of `StudentCourses`. Table represents a collection of entities, there is no need for plural names;

SM6 *Use of spaces for table names*, it is recommended to do not use spaces otherwise there will be the need of using {, [, " etc. characters to define tables and access data contained therein (e.g., use `StudentCourse` instead of `"Student Course"`);

SM7 *Use of unnecessary prefixes or suffixes for table names*, e.g., the name `School` is better than `TblSchool`, `SchoolTable`, etc.;

SM8 *`id` fields for tables are missed.* If `id` is not required for the time being, it may be required in the future (e.g., to support association tables, indexing, etc.);

SM9 *Use of varchar data types for indexing*. It is recommended to choose columns with the integer data type (or its variants) for indexing. Indexing based on strings might cause performance problems;

By considering such a list of smells we defined an extensible approach to implement and apply their detection on TyphonML models. Initial technical results were already presented in D2.4 [3]. For this deliverable, we consolidated the approach by implementing the smells previously overviewed and integrated the developed tools in the context of the process shown in Fig. 1. In particular, smell detection is performed once full TyphonML specifications are obtained. To this end, the requirement-aware approach presented in D3.3 [4] can be applied to augment the specification of the conceptual data entities with corresponding data-mappings.

3.2 Implementing and managing smell detectors

TyphonML modeling tools have been designed also to give early feedback about the specified models. In particular, models are analysed by a set of checks each devoted to the discovery of possible issues. Even though the analysis tools include already ready to use checks, it is possible to extend the system by specifying additional checks that modelers might want to add for the particular models at hand. In the context of the Typhon project, we expect that use case partners will ask the implementation of additional checkers as needed.

The conceived analysis tools have been developed by relying on Epsilon [1], which is a platform providing a consistent and interoperable task-specific languages. To enable TyphonML validations, we rely on Epsilon Object Language (EOL)[1] and Epsilon Validation Language (EVL)[2] provided by the Epsilon platform.

EOL is an imperative programming language for creating, querying, and modifying EMF models. The primary aim of that language is to provide a reusable set of common model management facilities, atop which task-specific languages can be implemented. However, EOL can also be used as a general-purpose stand-alone model management language for automating tasks that do not fall into the patterns targeted by task-specific languages.

EVL is a validation language built on top of EOL and provides a number of features such as support for detailed user feedback, constraint dependency management, semi-automatic transactional inconsistency resolution and (as it is based on EOL) access to multiple models of diverse metamodels and technologies. The aim of EVL is to contribute model validation capabilities to Epsilon. More specifically, EVL can be used to specify and evaluate constraints on models of arbitrary metamodels and modelling technologies. EVL also supports dependencies between constraints (e.g., if constraint A fails, the constraint B cannot be evaluated), customizable error messages to be displayed to the user and specification of fixes (in EOL) which users can invoke to repair inconsistencies. Also, as EVL builds on EOL, it can evaluate inter-model constraints (unlike OCL). Finally, the language permits to handle the severity of validation result:

- **Constraints**: they are used to capture critical errors that invalidate the model;
- **Critiques**: they are used to capture situations that do not invalidate the model, but should nevertheless be addressed by the user to enhance the quality of the model.

The detection of the smells defined in the previous section are already available in the TyphonML modeling supporting tools as a set of predefined EVL and EOL specifications, even though developers can possibly extend the set of available checks by creating new EVL and EOL files (or extending existing ones).

[1] `https://www.eclipse.org/epsilon/doc/eol/`
[2] `https://www.eclipse.org/epsilon/doc/evl/`

In the following we make an overview of the source code that has been developed to implement the detection of the smells previously presented and even to suggest fixes when possible. The complete implementation of the TyphonML tools including the smell detectors discussed in this document is available at `https://github.com/typhon-project/typhonml/`.

3.2.1 Preliminary checks

Before the detection of smells, preliminary checks are made to ensure the conformity of the conceptual model that is being defined through TyphonML. For example, the checker in Listing 2 checks if all the modeled entities have a name.

```
1  context TyphonML!NamedElement {
2            // Every NamedElement must define a name
3            constraint HasName {
4              check : self.name <> ""
5              message : "Element " + self + " must define a name"
6    }
7  }
```

Listing 2: EVL implementation of check if entity has a name.

3.2.2 SM1 detection: containment relations in relational databases

Listing 3 shows a fragment of the EVL specification implementing the detection of the smell *SM1*. The keyword **context** is used to specify the element type that has to be considered for evaluating the check. In case there are satisfied checks, a warning is triggered, and the related message is shown.

```
1  context Entity {
2            critique hasContainmentInER{
3                  check:
4                           not self.checkIsContainmentER()
5                  message: self.name+" entity, mapped in Relational Database, " +
6                                          "has a containment. Consider to move it to Document Database
                                               ↪ ."
7                  fix {
8                          title: "Fix Containment changing " +
9                                          " database from Relational to Document"
10                         do {
11                                 "Refactoring".println();
12                                 self.changeDBFromRelationalToDocument();
13                         }
14                 }
15           }
16 }
```

Listing 3: EVL check if ER type database has a containment

The actual implementation of the EVL check defined in List. 3 is instead implemented through some **operation** written in EOL (see lines 1, 5, 19 and 32 in List. 4). Specifically, the `checkIsContainmentER()` in line 1 is used by the **check** in line 3 and 4 in List. 3. This operation goes to call another one, the

`getAllIsContainmentER()` in line 5, which deals with the actual check on the entity being referred to and returns a list of entities that have the containment.

The other two operations (at line 19 and 32) are used instead in the fixing phase (see List. 3 at line 7). Indeed, the use of EVL allows defining fix operations, ie, modifying the model itself in order to correct, for example, a violation of a best practice.

```
1  operation Entity checkIsContainmentER() : Boolean{
2          return self.getAllIsContainmentER().notEmpty();
3  }
4
5  operation Entity getAllIsContainmentER() : List<Entity>{
6          var er_containmentEntities : List<Entity>;
7          for(rel in self.relations){
8                  if(rel.isContainment <> null and rel.isContainment == true){
9                          if(not TyphonML!Table.allInstances()->select(tab|tab.entity = rel.type).
                               ↪ isEmpty()){
10                                 (self.name + " has "+rel.type.name+" (with reference "+rel.name+") "
                                       ↪  +
11                                 "as containment and mapped in ER").println();
12                                 er_containmentEntities.add(rel.type);
13                         }
14                 }
15         }
16         return er_containmentEntities;
17 }
18
19 operation Entity changeDBFromRelationalToDocument(){
20         for(ent in self.getAllIsContainmentER()){
21                 for(db in self.eContainer().databases){
22                         if(db.isTypeOf(RelationalDB)){
23                                 var tableToDelete = db.tables.selectOne(tb : Table | tb.entity ==
                                       ↪ ent);
24                                 var nameOfTheDatabaseToTransfer := tableToDelete.name;
25                                 delete tableToDelete;
26                                 db.addCollectionToDocumentDB(nameOfTheDatabaseToTransfer, ent);
27                         }
28                 }
29         }
30 }
31
32 operation Database addCollectionToDocumentDB(inputName : String, ent : Entity){
33         for(db in self.eContainer().databases){
34                 if(db.isTypeOf(DocumentDB)){
35                         var newCollection := new TyphonML!‘Collection‘;
36                         newCollection.name = inputName;
37                         newCollection.entity = ent;
38                         db.collections.add(newCollection);
39                 }
40         }
41 }
```

Listing 4: EOL implementation of EVL `hasContainmentInER()` in line 4 of List. 3.

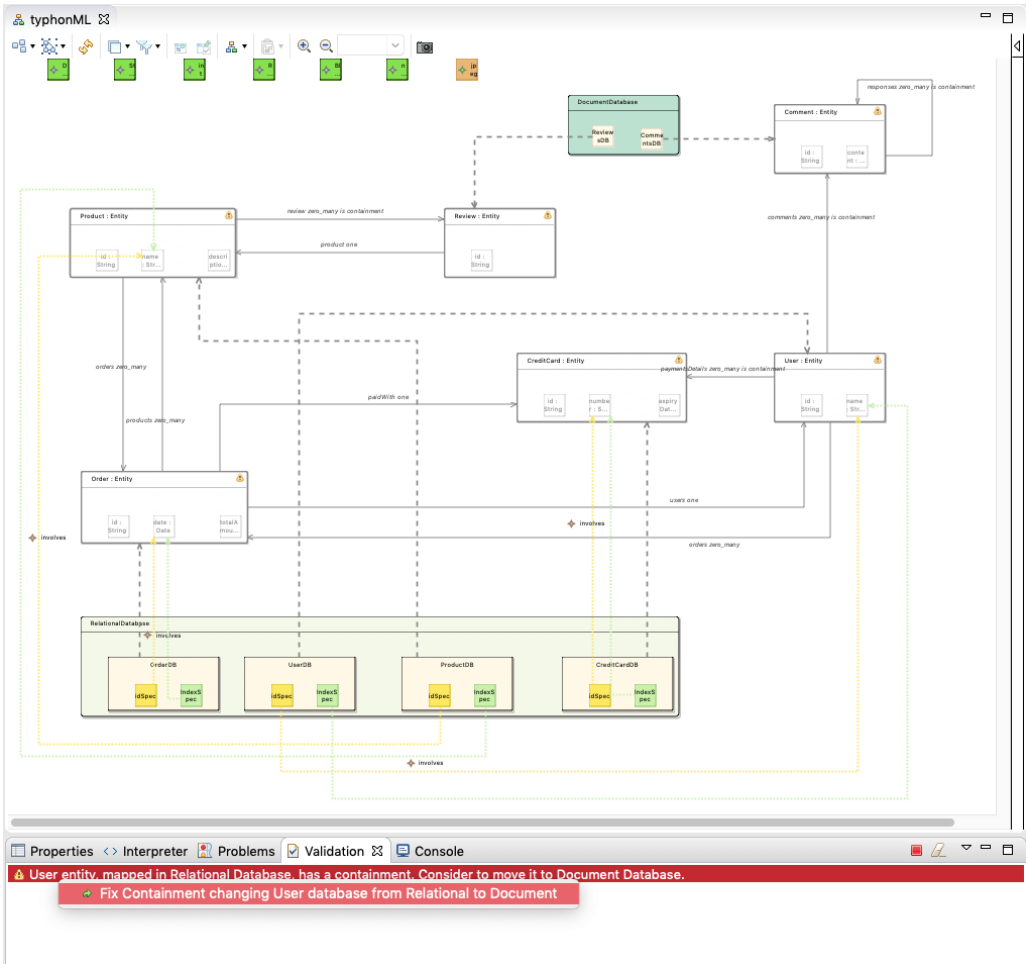

Figure 2: Fix Option from Eclipse Validation View

Further than detecting smells, checkers are also able to provide recommendations (when possible) about how to fix the detected smells. For instance, Fig. 2 shows a warning related to a matched entity affected by the smell *Containement relations in relational databases* and gives the possibility to apply a fixing procedure to solve the problem. One way to solve the issue would be to change the mapping of the considered entity from relational to document database.

3.2.3 SM2 detection: God entity

The implementation of the smell *SM2* is shown in Listing 5. In particular, for each entity the operation `checkGodTable` is executed in order to check if the number of contained attributes is higher than the `maxNumberOfAttributes` (which can be pre-configured as needed).

```
1 operation Entity checkGodTable(maxNumberOfAttributes : Integer) : Boolean{
2         if(self.attributes.size > maxNumberOfAttributes){
3                 return true;
4         }else{
5                 return false;
6         }
7 }
```

Listing 5: EOL implementation of SM2 detector.

3.2.4 SM3 detection: Overloaded attribute name

Listing 6 shows the implementation of the checker for smell SM3. The operation is able to retrieve all the attributes, belonging to different entities, with the same name and different types.

```
operation Entity checkOverloadedAttributeName() : String{
        //delete self entity from the list of all entities
        var filteredEntities = TyphonML!Entity.allInstances()->select(e|e <> self);
        for(selfAttribute in self.attributes){
                for(entity in filteredEntities){
                        for(attribute in entity.attributes){
                                if(selfAttribute.name == attribute.name){
                                        if(selfAttribute.type.name <> attribute.type.name){
                                                return selfAttribute.name;
                                        }
                                }
                        }
                }
        }
        return null;
}
```

Listing 6: EOL implementation of SM3 detector.

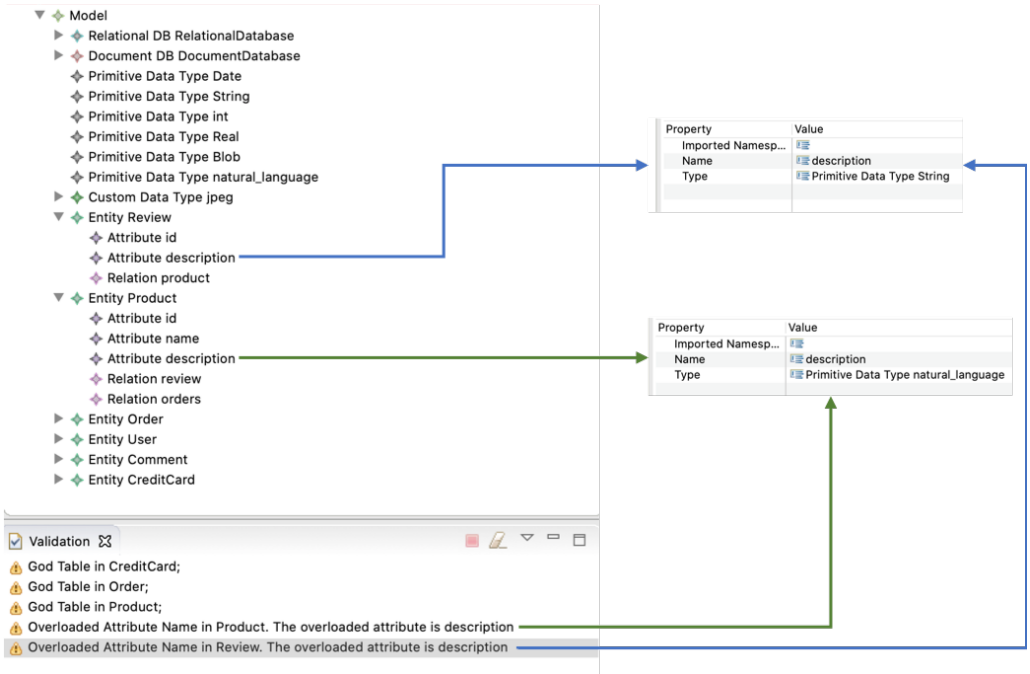

Figure 3: Explanatory application of the SM2 and SM3 detectors

Figure 3 shows the result of the application of SM2 and SM3 on the TyphonML model given in Listing 1. Specifically, the *god entities* `CreditCard`, `Order`, and `Product` have been detected[3]. Additionally, also an

[3]It must be taken into account that the threshold that establishes when a got entity is detected is parametric and it can be set by the user. In the example, the considered threshold is purposely set low i.e., 2.

overloaded attribute name has been detected together with the corresponding involved entities. In particular, the attribute `description` has been defined in the entities `Review` and `Product` with the same name but with different `Type` (i.e., `String` and `natural_language`, respectively).

3.2.5 SM4 detection: Meaningless name

The implementation of SM4 requires the use of tools that make it possible to detect aspects that are not only syntactic but also semantic. Smell detection of this type requires the use of tools that can process natural languages so that information about words and their meaning can be extracted. This information is traditionally provided through dictionaries, and for our goal, we choose to use WordNet[2] that is a lexical database for the English language. WordNet provides a more effective combination of traditional lexicographic information and modern computing.

In order to use the potential of WordNet, it had to be integrated into the Epsilon framework. To do this, we have created an extensible ad-hoc plug-in in the sense that the methods that execute checks making use the potential of WordNet can be extended, and modified.

Listing 7 shows the Java class of the WordNet utility for TyphonML smell detection. Specifically in lines 12-23 there is the method that has to be called to check if a word exists in the WordNet database.

```
1  public WordNet() {
2                  try {
3                          JWNL.initialize(new FileInputStream(getFilePath(getBundle().getResource(
                               ↪ JWNL_CONFIGURATION_FILEPATH).getPath())));
4                  } catch (FileNotFoundException e) {
5                          e.printStackTrace();
6                  } catch (JWNLException e) {
7                          e.printStackTrace();
8                  }
9                  wordNetDictionary = Dictionary.getInstance();
10         }
11
12         public boolean checkifWordExists(String r) {
13                 boolean flag = false;
14                 try {
15                         IndexWord word = wordNetDictionary.getIndexWord(NOUN, r);
16                         if (word != null) {
17                                 flag = true;
18                         }
19                 } catch (JWNLException ex) {
20                         return false;
21                 }
22                 return flag;
23         }
24
25         public boolean checkIfWordPlural(String r) {
26             if( _nounToLemma == null ) _nounToLemma = new HashMap<String, String>();
27
28             // save time with a table lookup
29             if( _nounToLemma.containsKey(r) ) return true;
30
31             try {
32               // don't return lemmas for hyphenated words
```

```
33              if( r.indexOf('-') > -1 || r.indexOf('/') > -1 ) {
34                _nounToLemma.put(r, null);
35                return false;
36              }
37
38              // get the lemma
39              IndexWord iword = Dictionary.getInstance().lookupIndexWord(POS.NOUN, r);
40              if( iword == null ) {
41                _nounToLemma.put(r, null);
42                return false;
43              }
44              else {
45                String lemma = iword.getLemma();
46
47                if( r.equals(lemma) ) {
48                  // Some nouns have their plural in WordNet as a strange rare word (e.g. devices).
49                  // Here we guess the single form, and return it if the guess exists (e.g. device).
50                  if( r.endsWith("es") ) {
51                    String guess = r.substring(0, r.length()-1);
52                    IndexWord iGuess = Dictionary.getInstance().lookupIndexWord(POS.NOUN, guess);
53                    if( iGuess != null && guess.equals(iGuess.getLemma()) ) {
54                      lemma = guess;
55                    }
56                  }
57
58                  // "men" and "businessmen" are in WordNet as lemmas ... we need to get the
                        ↪ singular man
59                  else if( r.endsWith("men") ) {
60                    String guess = r.substring(0, r.length()-2) + "an";
61                    IndexWord iGuess = Dictionary.getInstance().lookupIndexWord(POS.NOUN, guess);
62                    if( iGuess != null && guess.equals(iGuess.getLemma()) ) {
63                      lemma = guess;
64                    }
65                  }
66
67                  else if( r.equals("people") )
68                    return true;
69                }
70
71                if( lemma.indexOf(' ') != -1 ) // Sometimes it returns a two word phrase
72                  lemma = lemma.trim().replace(' ','_');
73
74                _nounToLemma.put(r, lemma);
75                return true;
76              }
77            } catch( Exception ex ) { ex.printStackTrace(); }
78
79            return false;
80          }
81
82          public boolean isPrefix(String s1, String s2) {
83                StringBuilder sb = new StringBuilder();
84                int count = 1;
```

```
85                  for (int i = 0; i < s2.length() ; i++) {
86                          sb.append(s2.charAt(i));
87                          if(count > 1) {
88                                  if(s1.startsWith(sb.toString())) {
89                                          return true;
90                                  }
91                          }
92                          count++;
93                  }
94                  return false;
95          }
96
97          public boolean isSuffix(String s1, String s2) {
98                  String s1Reverse = null;
99                  for(int i = s1.length() - 1; i >= 0; i--){
100                         s1Reverse = s1Reverse + s1.charAt(i);
101         }
102
103                 StringBuilder sb = new StringBuilder();
104                 int count = 1;
105                 for (int i = s2.length()-1; i >=0 ; i--) {
106                         sb.append(s2.charAt(i));
107                         if(count > 1) {
108                                 if(s1Reverse.startsWith(sb.toString())){
109                                         return true;
110                                 }
111                         }
112                         count++;
113                 }
114                 return false;
115         }
116
117         private String getFilePath(String path) {
118                 Bundle bundle = getBundle();
119                 URL fileURL = bundle.getResource(path);
120                 String finalPath = "";
121                 try {
122                         finalPath = FileLocator.resolve(fileURL).getFile();
123                         return java.nio.file.Paths.get(finalPath).normalize().toString();
124                 } catch (IOException e) {
125                         // TODO Auto-generated catch block
126                         e.printStackTrace();
127                 }
128                 return finalPath;
129         }
130
131         private Bundle getBundle() {
132                 //return Platform.getBundle("TyphonMLWordNetUtility");
133                 return FrameworkUtil.getBundle(getClass());
134         }
135 }
```

Listing 7: TyphonML WordNet java plug-in class for SM4.

This method is called by the dedicated EOL operation that, in turn, is called from a dedicated EVL check (see Listing 8 line 3). Specifically, in line 2 the EOL code that permits to call external java classes is executed. In line 3 and 7 it is called the actual method (see lines 12-23 in Listing 7).

```
1  critique hasMeaninglessNames{
2                  check{
3                                  meaningName = self.checkMeaninglessName();
4                                  if(meaningName <> null){
5                                          return false;
6                                  }else{
7                                          return true;
8                                  }
9                          }
10                 message: "Meaningless entity name or attributes in " + self.name + ". ["+meaningName
                       ↪ +"]."
11          }
```

Listing 8: EVL implementation of the SM4 detector.

Once the methods necessary to detect the smell have been defined, the plug-in is imported and, therefore, integrated into the EVL evaluator system.

```
1  operation Entity checkMeaninglessName(): String{
2          var wordNet = new Native("typhonmlwordnetutility.WordNet");
3          if(not wordNet.checkifWordExists(self.name)){
4                  return self.name;
5          }else{
6                  for(attr in self.attributes){
7                          if(not wordNet.checkifWordExists(attr.name)){
8                                  return attr.name;
9                          }
10                 }
11         }
12         return null;
13 }
```

Listing 9: EOL implementation of SM4.

Figure 4 shows the output of a sample application of the SM4 detector. In particular, the shown wondow dialog lists the warning in which for example one entity was called erroneously *UUser*. Thus, in that way the user can detect also possible typos in entity names (like *UUser*) and attributes names (like *description123* attribute in the shown *Review* entity). The system also provides information on which attribute of which entity has the

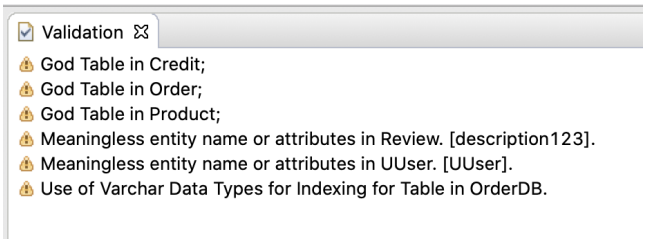

Figure 4: Explanatory application of the SM4 detector.

name without meaning (see name in square brackets in the Fig. 4).

3.2.6 SM5 detection: Use of plural for table names

As described for the SM4, also for this SM5 we used WordNet going to recall the EOL operation (see List. 10) through the EVL code (see 11) which in turn calls the code of the external Java method, or that in line 25-80 of Listing 7.

```
operation Entity checkPluralName(): String{
        var wordNet = new Native("typhonmlwordnetutility.WordNet");
        if(not wordNet.checkIfWordPlural(self.name)){
                return self.name;
        }else{
                for(attr in self.attributes){
                        if(not wordNet.checkIfWordPlural(attr.name)){
                                return attr.name;
                        }
                }
        }
        return null;
}
```

Listing 10: EOL implementation of SM5.

```
//PLURAL NAMES Smell
critique hasPluralNames{
        check{
                        pluralName = self.checkPluralName();
                        if(pluralName <> null){
                                return false;
                        }else{
                                return true;
                        }
                }
        message: "Use of plural entity name or attributes in " + self.name + ". ["+pluralName+"]."
         }
```

Listing 11: EVL implementation of the SM5 detector.

3.2.7 SM6 detection: Use of spaces for table names

The implementation of SM6 is shown in Listing 12. The operation is able to detect all the entity and attribute names containing white spaces.

```
operation Entity checkUseSpacesforTableNames(): String{
        if(" ".isSubstringOf(self.name)){
                self.name.println();
                return self.name;
        }else{
                for(attr in self.attributes){
                        if(" ".isSubstringOf(attr.name)){
                                attr.name.println();
                                return attr.name;
                        }
```

```
11                }
12        }
13        return null;
14 }
```

Listing 12: EOL implementation of SM6.

Figure 5 shows the result of an explanatory application of the checker on the running example.

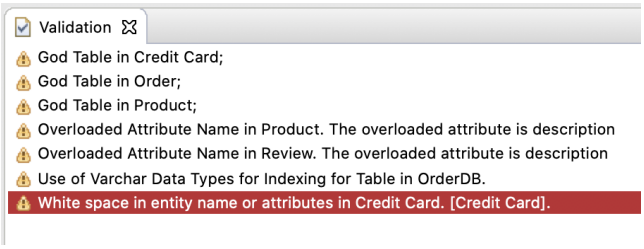

Figure 5: Explanatory application of the SM6 detector.

3.2.8 SM7 detection: Use of unnecessary prefix or suffixes for table Names

The implementation of the SM8 detector is shown in Listing 13

```
1  critique hasUseOfUnnecessaryPrefixOrSuffixForTableNames{
2                 check{
3                                 suffixPrefixName = self.
                                      ↪ checkUseOfUnnecessaryPrefixOrSuffixForTableNames();
4                                 if(suffixPrefixName <> null){
5                                         return false;
6                                 }else{
7                                         return true;
8                                 }
9                         }
10                message: "The entity " + self.name + " has a prefix or suffix or another ["+
                      ↪ suffixPrefixName+"]."
11          }
```

Listing 13: EVL implementation of the SM7 detector.

In the EOL code, the operation that is called by the EVL code (see line 3 of Listing 13) is presented in Listing 14.

```
1  operation Entity checkUseOfUnnecessaryPrefixOrSuffixForTableNames(): String{
2      var wordNet = new Native("typhonmlwordnetutility.WordNet");
3      var filteredEntities = TyphonML!Entity.allInstances()->select(e|e <> self);
4      //For all the other entities different for sel check the name suffix
5      for(s in filteredEntities){
6              if(wordNet.isSuffix(self.name, s.name) or wordNet.isPrefix(self.name, s.name)){
7                         return self.name;
8                 }
9      }
10         return null;
11 }
```

Listing 14: EOL implementation of SM7.

In Fig. 6 the result of a sample application of such a checker is shown and it consists of two entities sharing the same suffix (*123_*).

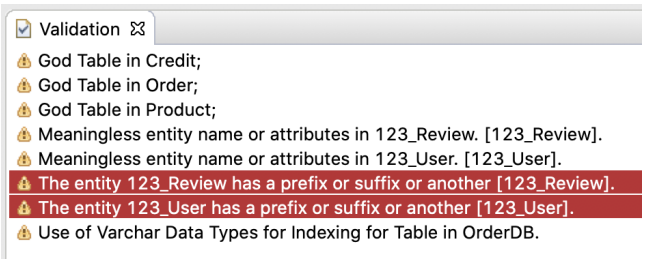

Figure 6: Explanatory application of the SM7 detector.

3.2.9 SM8 detection: ID fields for tables are missed

The implementation of the SM8 detector is shown in Listing 15. In particular, the detector makes sense for relational databases and thus for the tables containing `idSpec` elements. The operation in Listing 15 is used in the EVL implementation part shown in Listing 16. In particular, the EVL code induces the visualization of window dialogs like that shown in Fig. 7 showing all the tables that are affected by the smell SM8: in the shown example the entity `User` has been mapped to the relational table `UserDB` and no ids have been specified for it.

```
operation TyphonML!Table checkIDFieldsForTablesAreMissing() : Boolean{
        if(self.idSpec <> null){
                if(self.idSpec.attributes <> null){
                        return true;
                }
        }
        return false;
}
```

Listing 15: EOL implementation of the SM8 detector.

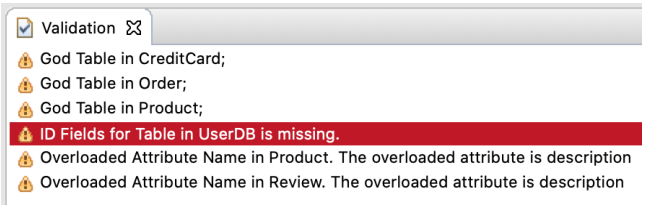

Figure 7: Explanatory application of the SM8 detector.

```
context Table {
         //ID Fileds For Tables Are Missing Smell
         critique hasIDFieldsForTablesAreMissingTable{
                check: self.checkIDFieldsForTablesAreMissing()
                message: "ID Fields for Table in " + self.name +" is missing."
         }
}
```

Listing 16: EVL implementation of the SM8 detector.

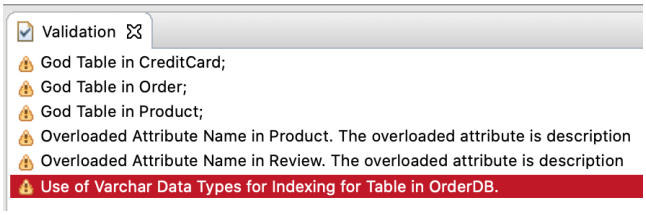

Figure 8: Explanatory application of the SM9 detector.

3.2.10 SM9 detection: Use of varchar data types for indexing

The implementation of the SM9 detector is shown in Listing 17. The operation is used by the EVL code shown in Listing 18 which shows window dialogs like that shown in Fig. 8. In the shown example, the entity `Order`, which is mapped to a relational database, has as index (`indexSpec` in the TyphonML metamodel) an attribute of type *Varchar*.

```
operation TyphonML!Table checkUseOfVarcharDataTypesForIndexing() : Boolean{
	if(self.indexSpec <> null){
		if(self.indexSpec.attributes <> null){
			for(attr in self.indexSpec.attributes){
				if(attr.type.name.toLowerCase() == "string"){
					return true;
				}
			}
		}
	}
	return false;
}
```

Listing 17: EOL implementation of the SM9 detector.

```
context Table {
	...
	//Use of Varchar Data Types for Indexing Smell
	critique hasUseOfVarcharDataTypesForIndexing{
		check: self.checkUseOfVarcharDataTypesForIndexing()
		message: "Use of Varchar Data Types for Indexing for Table in " +
		self.name +"."
	}
}
```

Listing 18: EVL implementation of the SM9 detector.

4 Conclusions

In this document we focused on the notion of TyphonML smells. We proposed a tool supported approach to detect TyphonML elements that might indicate the violation of the recommended best practices and potentially affect in negative way the quality of the modeled system. The smell detection phase is located in a more general analysis and reasoning process, which involves additional steps aiming at guiding the development of polystores satisfying functional and non-functional requirements explicitly specified by the developer.

For future work we will continue the collaboration with the TYPHON partners especially to satisfy the requirements and needs of the use case partners. Moreover, even though this document is supposed to be the last deliverable of WP2, we will continue with the development and enhancement of TyphonML supporting tools for properly addressing requests for improvement coming from the different tool users.

References

[1] Dimitrios Kolovos, Louis Rose, Richard Paige, and Antonio Garcıa-Domınguez. The epsilon book. *Structure*, 178:1–10, 2010.

[2] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[3] The University of L'Aquila. D2.4 – TyphonML Modelling Tools, 2019.

[4] The University of L'Aquila. D3.3 – TyphonML to TyphonDL Model Transformation Tools, 2019.

[5] Tushar Sharma, Marios Fragkoulis, Stamatia Rizou, Magiel Bruntink, and Diomidis Spinellis. Smelly relations: measuring and understanding database schema quality. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 55–64. ACM, 2018.