Project Number 780251

D2.4 TyphonML Modelling Tools

Version 1.0
28 June 2019
Final

Public Distribution

University of L'Aquila

Project Partners: Alpha Bank, ATB, Centrum Wiskunde & Informatica, CLMS, Edge Hill University, GMV, Nea Odos, 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
Nea Odos Charalampos Daskalakis Themistocleous 87 106 83 Athens Greece Tel: +30 210 344 7300 E-mail: cdaskalakis@neaodos.gr	**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	6 March 2019
0.2	First draft	1 June 2019
0.7	First full draft	21 June 2019
0.8	Further editing draft	26 June 2019
1.0	Final updates after partner reviews	28 June 2019

Table of Contents

Executive Summary

In recent years, NoSQL databases have emerged as an alternative approach to data storage, event though they still remain far from the level of maturity of relational databases. While there is some work towards this direction, the proposed solutions are technology-specific and not applicable across different classes of NoSQL data stores.

This document presents the supporting tools that have been developed in the context of the WP2 of the TYPHON project to support the specification of TyphonML models together with their evolution. The tools permit developers to specify conceptual entities by abstracting the specificities of the underlying technologies. Mappings to specific database systems are enabled by interestingly giving the possibility of managing polystores consisting of both relational and NoSQL databases. Further than presenting an overview of the developed foundations, the document contains concrete examples that users can follow to actually use the proposed TyphonML modeling tools.

1 Introduction

The aim of TYPHON is to provide an industry-validated methodology and integrated technical offering for designing, developing, querying, evolving, analysing and monitoring architectures for scalable persistence of hybrid data (relational, graph-based, document-based, textual etc.).

In the context of TYPHON, WP2 is focusing on the development of languages and tools for *designing* hybrid polystores taking into account the structure of the data and the available deployment resources. In particular, the main objectives of WP2 are the development of the TyphonML to model in a homogeneous manner data that need to be stored in polystores. In the previous deliverables, the TyphonML language has been designed and validated with respect to requirements that have been elicited together with the industrial partners of the project. In this document, the modeling tools supporting the specification and usage of TyphonML models are presented. In particular, we present results of WP2 related to the following task (from the TYPHON DoW):

> *Task 2.3: TyphonML Modelling Tools Design and Implementation.* This task will design and develop modelling tools that engineers can use to define TyphonML models. The developed modelling tools will feature different viewpoints (data, requirements, infrastructure-centric) that will enable engineers to model hybrid polystores at a high level of abstraction. We anticipate that several different concrete syntaxes (diagram-based, text-based, table-based etc.) may be necessary to accommodate for the particularities of each view point. TyphonML will be developed by exploiting mature technologies available in the Eclipse ecosystem. For the specification of the abstract syntax of the language, the Ecore/EMF metamodelling stack will be used. The development of the textual concrete syntax can rely on technologies like EMFText and XText, while the graphical editors for TyphonML models can be developed using technologies like GMF, and Eugenia. TyphonML will be also implemented as a set of UML profiles to enable wider adoption in the industry.

As presented in deliverable D2.3 [2], TyphonML has been designed so to enable the specification and analysis of data to be managed and stored on hybrid polystores. TyphonML models play a key role in the overall Typhon vision since they underpin several phases of polystore life-cycle including deployment, query, and evolution as shown in Fig. 1.

TyphonML specifications logically consist of four main types of elements i.e., definition of data types, specification of conceptual entities and relationships, mappings of conceptual elements on concrete database technologies. Moreover, in case the considered TyphonML model has been already deployed, the language permits modelers to specify changes that can be operated so to enable the automatic migration of the already stored data and make them consistent with the data schema being specified in the new version of the TyphonML model.

This document presents the tools that have been developed in the context of WP2 to support the specification of TyphonML models, their analysis and evolution. According to the use case requirements described in D1.1 [5], both textual and graphical editors have been developed. Moreover, validation tools have been also provided to enable early analysis of TyphonML models.

1.1 Structure of the deliverable

The structure of the deliverable is as follows: the TyphonML textual editor is presented in Section 2, whereas the graphical one is the subject of Section 3. Section 4 presents the change operators that are available to specify

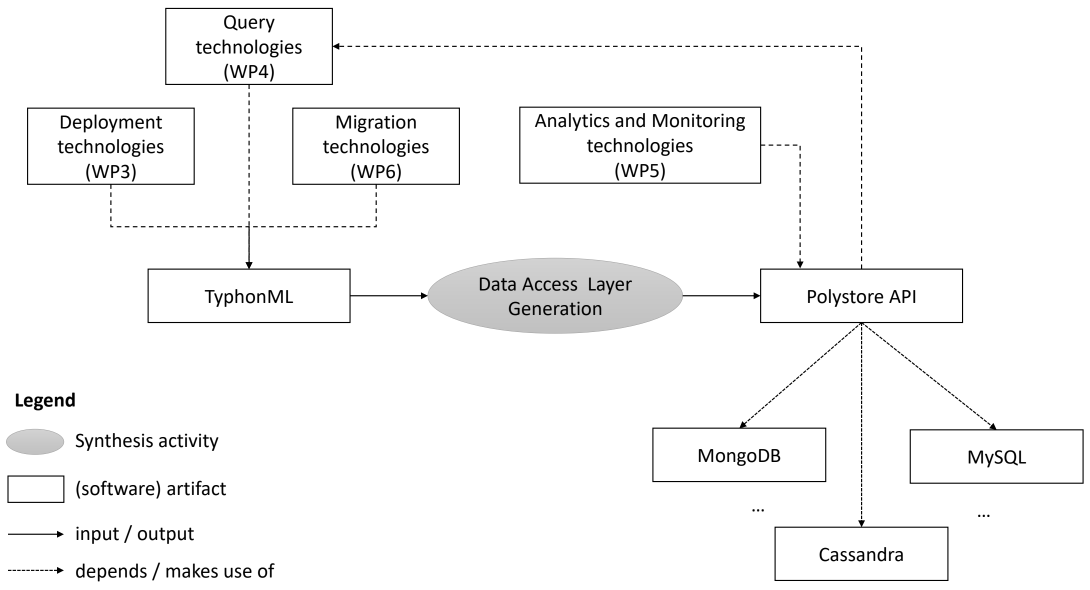


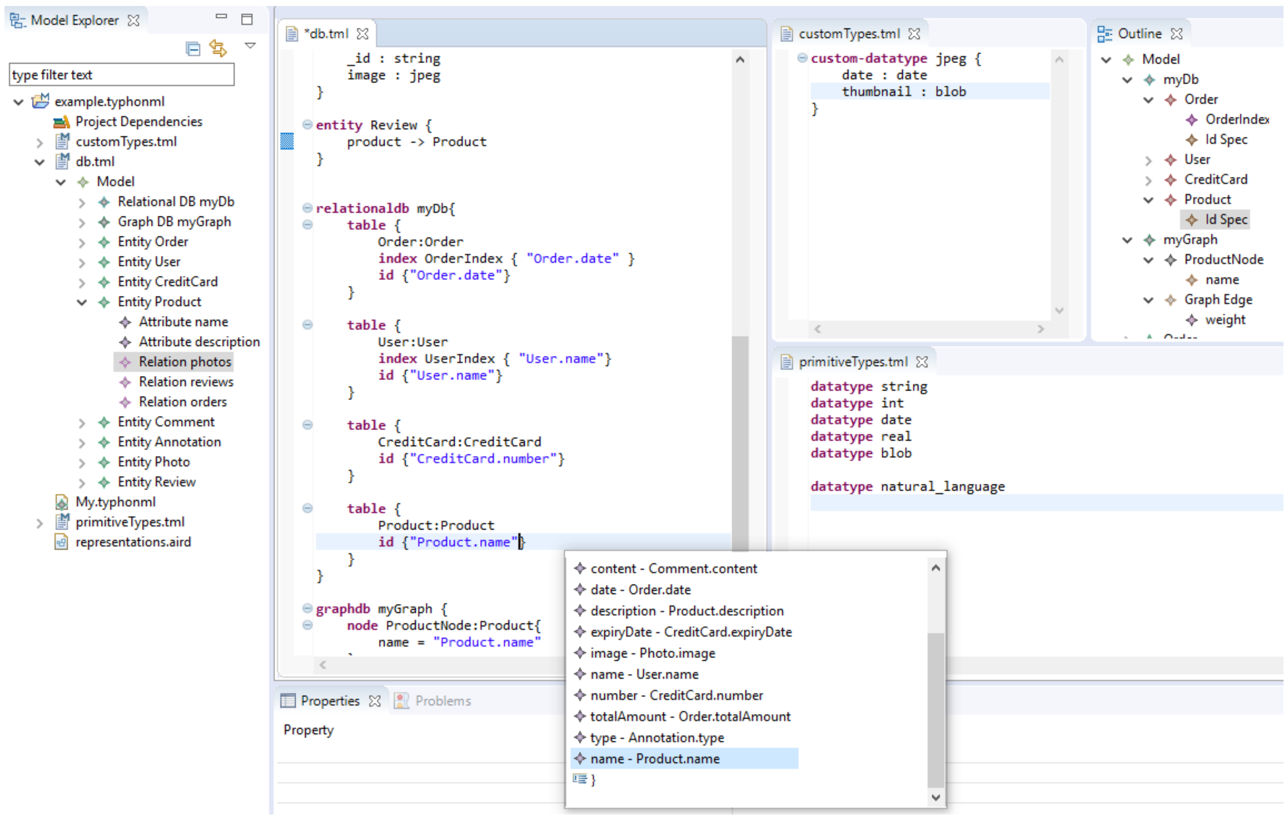
Figure 1: TyphonML in the overall TYPHON picture (from D2.3 [2])

how existing TyphonML specifications need to be evolved. The validation facilities that permit to perform early analysis of TyphonML models are presented in Section 5. Section 6 presents the code generator for producing the data access layer consisting of an API that developers can programmatically use for performing CRUD operations on the modeled systems. Section 7 concludes the document and provides an overview of the next steps.

The document has been defined so to provide the reader with enough details that permit users to play with the developed modeling tools, which are publicly available at `https://github.com/typhon-project/typhonml`. To this end, Appendix A is given in order to present the technical dependences that need to be satisfied for using the TyphonML modeling tools. A guideline to install such dependences is also presented.

2 The TyphonML textual editor

The TyphonML textual editor has been developed by means of Xtext[1], which is an Eclipse project for developing domain-specific languages. Starting from the specification of the grammar of the language, the Xtext framework supports the implementation of a full infrastructure, including parser, linker, type-checker, compiler as well as editing support for Eclipse. The developed TyphonML textual editor provides modelers with typically expected features like syntax highlighting, code completion, and outlines.



```
    _id : string
    image : jpeg
}

entity Review {
    product -> Product
}


relationaldb myDb{
    table {
        Order:Order
        index OrderIndex { "Order.date" }
        id {"Order.date"}
    }

    table {
        User:User
        index UserIndex { "User.name"}
        id {"User.name"}
    }

    table {
        CreditCard:CreditCard
        id {"CreditCard.number"}
    }

    table {
        Product:Product
        id {"Product.name"}
    }
}

graphdb myGraph {
    node ProductNode:Product{
        name = "Product.name"
```

```
custom-datatype jpeg {
    date : date
    thumbnail : blob
}
```

```
datatype string
datatype int
datatype date
datatype real
datatype blob

datatype natural_language
```

```
content - Comment.content
date - Order.date
description - Product.description
expiryDate - CreditCard.expiryDate
image - Photo.image
name - User.name
number - CreditCard.number
totalAmount - Order.totalAmount
type - Annotation.type
name - Product.name
}
```

Figure 2: The TyphonML textual editor at work (to be replaced)

Figure 2 shows the TyphonML editor at work. Custom and primitive data types can be separately defined (in order to enable their reuse in different projects) from the actual specification of the data entities to be stored. In the following, the different constructs that are available in the TyphonML textual language are separately presented.

2.1 Datatypes

Further than primitive data types (e.g., string, data, integer, real, etc.), the language permits modeler to define new types by means of the keyword **`customdatatype`**. For instance, developer can define a new datatype named `Jpeg` consisting of three elements i.e., `date`, `thumbnail` and `content` for enabling the storage of the creation date, the thumbnail, and the actual content, respectively of `jpeg` pictures being stored. The items of custom

[1] https://www.eclipse.org/Xtext/

datatypes can be either primitive (e.g., Date) or defined in some package to be referred as in the case of thumbnail and content elements in Listing 1 that are of type Blob defined in the package `it.univaq.disim.Blob` (see lines 4–5 in Listing 1).

```
1  customdatatype Jpeg {
2          elements{
3                  date : Date,
4                  thumbnail : Blob ["it.univaq.disim.Blob"],
5                  content: Blob ["it.univaq.disim.Blob"]
6          }
7  }
```

Listing 1: Definition of custom datatypes

2.1.1 Defining conceptual entities

After the definition of custom data types, modelers can finally start with the definition of the concepts and relationships that will be managed by the information systems being developed. Conceptual entities can be defined by means of the keyword **`entity`** as shown in Listing 2. Each entity specification includes the definition of contained attributes and references. In particular, attributes are defined with

```
<name> ':'  <PrimitiveType|CustomDataType>
```

elements, whereas relationships occurring between entities are defined by means of

```
<name> '->' <Entity>[<Cardinality>]
```

elements. It is also possible to specify if the target entity is contained (e.g., to trigger cascade-deletion) by using ':->' instead of '->'. For instance, Listing 2 contains the specification of the conceptual entities underpinning a simple e-commerce system. The defined `Product` entity defined therein, consists of the attributes `name` and `description` of type `String`. Moreover, each product can contain several reviews (see the containment reference with target entity `Review` at line 8 of Listing 2) and can be an item contained in possibly different orders (see line 9). It is important to remark that when defining a reference from one entity (e.g., named `Product`) to a second entity (e.g., named `Review`) it is possible to specify the opposite reference from `Review` to `Product` in order to define a bidirectional relation instead of two different unidirectional ones.

```
1  entity Review{
2          product -> Product[1]
3  }
4
5  entity Product{
6          name : String
7          description : String
8          review :-> Review."Review.product"[0..*]
9          orders -> Order[0..*]
10         photo : Jpeg;
11 }
12
13 entity Order{
14         date : Date
15          totalAmount : Int
16          products -> Product."Product.orders"[0..*]
17          users -> User."User.orders"[1]
18          paidWith -> CreditCard[1]
19  }
20
21  entity User{
22          name : String
23          surname : String
24          comments :-> Comment[0..*]
25          paymentsDetails :-> CreditCard[0..*]
26          orders -> Order[0..*]
27  }
28
29  entity Comment{
30          freetext content [SentenceSegmentation,TextClassification]
31          responses :-> Comment[0..*]
32  }
33
34  entity CreditCard{
35          number : String
36          expiryDate : Date
37  }
```

Listing 2: Definition of conceptual entities

2.2 Enabling natural language processing

In order to enable advanced text analysis, Edge Hill University (EHU) has conceived natural language processing tasks [4] that can be enabled from the TyphonML editor on all the conceptual attributes that are defined by means of the keyword **`freetext`** as for instance done for the attribute `content` of the `Comment` conceptual entity specified at line 30 of Listing 2.

Details about the analysis and the management of **`freetext`** attributes are given in D2.2. For the sake of this document, it is enough to mention that an Elasticsearch[2] back-end is configured to enable the application of natural language processing tasks as specified in the TyphonML models.

2.3 Mapping conceptual models to relational databases

The conceptual entities defined as previously shown need to be mapped to actual database technologies. It's up to the modeler deciding how to allocate the modeled entities to the available databases. One of the database kinds supported by TyphonML is the relational one. Thus, the TyphonML editor provides the modelers with the constructs that are necessary to specify which entity should be stored in a relational database and how. To this end, by means of the keyword **`table`** modelers will define the tables that are needed to store the entities of interests. In particular, by referring to the example given in Listing 3, a table definition is given by specifying *i)* its name, *ii)* the reference to the conceptual entity that will be stored in the table being defined (e.g., the entity `Order` at line 4), *iii)* optionally the attributes of the referred entity that should be indexed to improve the

[2]https://www.elastic.co/

performance of the queries to be evaluated on the relational database being developed (see e.g., the attribute `date` of the entity `Order` as specified in line 6)).

```
 1 relationaldb RelationalDatabase{
 2         tables{
 3                 table {
 4                         OrderDB : Order
 5                         index orderIndex {
 6                                 attributes ("Order.date")
 7                         }
 8                 }
 9                 table {
10                         UserDB : User
11                         index  userIndex{
12                                 attributes ("User.name")
13                         }
14                 }
15                 table {
16                         ProductDB : Product
17                         index productIndex{
18                                 attributes ("Product.name")
19                         }
20                 }
21                 table {
22                         CreditCardDB : CreditCard
23                         index creditCardIndex{
24                                 attributes ("CreditCard.number")
25                         }
26                 }
27         }
28 }
```

Listing 3: Definition of conceptual entities

2.4 Mapping conceptual models to document databases

Listing 4 shows the specification of a simple document database storing reviews and comments related to products managed by the considered e-commerce system. The keyword **`collections`** permits to specify the collections that need to be stored. According to the specification of `ReviewCommentDB` given in Listing 4, the collections `ReviewsCol` and `CommentsCol` will store objects consisting of all the attributes defined in the `Review` and `Comment` entities defined in Listing 2, respectively.

```
documentdb ReviewCommentDB{
        collections{
                ReviewsCol : Review
                CommentsCol : Comment
        }
}
```

Listing 4: Definition of a simple document-based database

2.5 Mapping conceptual models to graph databases

TyphonML permits modelers to specify also data that need to be stored in a graph-like structure. For instance, Listing 5 shows the specification of a simple graph-based DB storing the concordance level among different `Products`. In particular, the defined `ConcordanceDB` consists of nodes referring the conceptual `Product` entity (see line 3), and for each of them only the corresponding `name` attribute is stored. Pairs of products are linked by edges, each representing the corresponding concordance value (e.g., see the edge `concordance` defined at line 8).

More in general, the construct **`node`** permits modelers to specify the nodes of the data structure being modeled. Each node consists of the reference to the conceptual entity of interest, and of attributes to be stored in each node of the graph. The keyword **`edge`** consists of the structural features that are needed to specify the source and target nodes of the edges being specified. Labels can be defined by means of the keyword **`labels`**. Each label is a named element and consists of the corresponding type that can be primitive, custom, or even an entity type.

```
 1  graphdb ConcordanceDB  {
 2          nodes {
 3                  node ProductNode!Product {
 4                          name = "Product.name"
 5                  }
 6          }
 7          edges {
 8                  edge concordance {
 9                          from ProductNode
10                          to ProductNode
11                          labels {
12                                  weight:int
13                          }
14                  }
15          }
16  }
```

Listing 5: Definition of a simple graph-based database

2.6 Mapping conceptual models to key-value databases

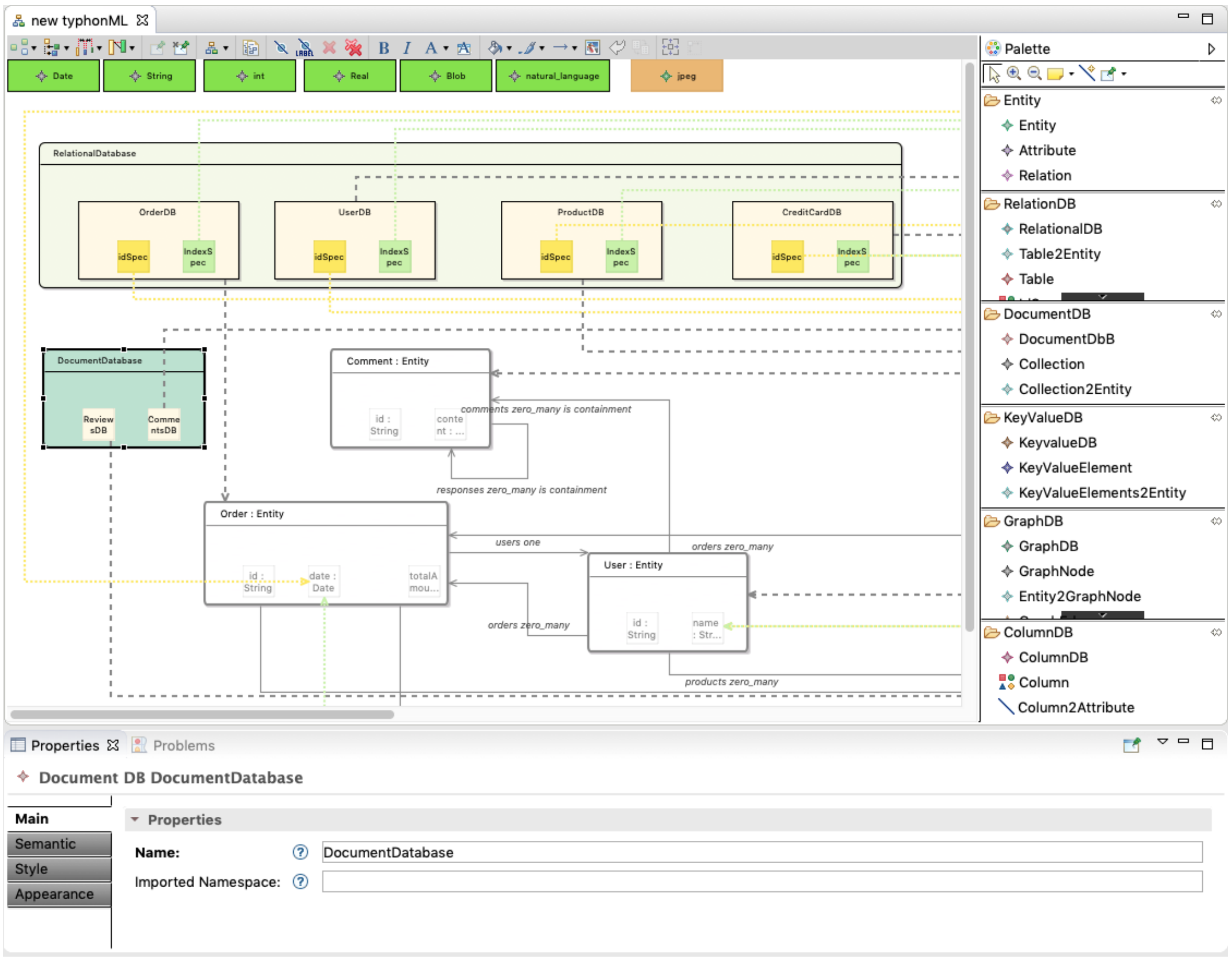
Key-value stores consist of sets of key-value pairs with unique keys. TyphonML permits modelers to specify aggregation of elements to be stored in key-value pairs by means of **`keyvaluedb`** elements as shown in Listing 6. The modeled `PhotoDB` permits to store elements referring to the `photo` attribute of the `Product` entity specified in Listing 2. Each element can be an aggregation of heterogeneous data even related to different conceptual entities.

```
keyvaluedb PhotoDB {
        elements {
                photocontent { photokey -> ( "Product.photo" )  }
        }
}
```

Listing 6: Definition of a simple key-value database

3 The TyphonML graphical editor

The TyphonML graphical editor is developed by relying on Sirius[3], which is an Eclipse project supporting the development of graphical modeling workbenches by leveraging the Eclipse Modeling technologies, including EMF and GMF. A typical workbench developed with Sirius is composed of a set of Eclipse editors (diagrams, tables, and trees) which allow the users to create, edit and visualize EMF models. The editors are defined by a model, which defines the complete structure of the modeling workbench, its behaviour, and all the editing and navigation tools. A runtime dynamically interprets the description of a Sirius modeling workbench within the Eclipse IDE. For supporting the specific need for customization, Sirius is extensible in many ways, notably by providing new kinds of representations, new query languages and by being able to call Java code to interact with Eclipse or any other system.



Document DB DocumentDatabase

Properties

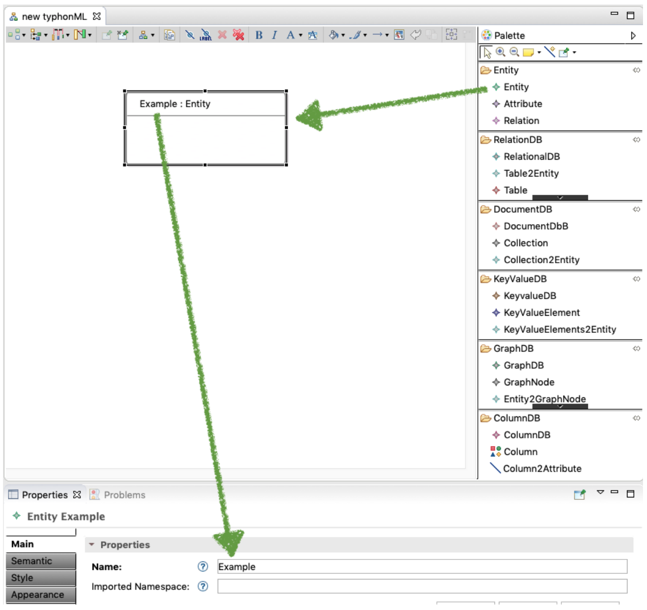
Name: DocumentDatabase

Imported Namespace:

Figure 3: The TyphonML graphical editor at work

The TyphonML graphical editor comes with a palette containing tools allowing users to create new model elements. Specifically, the palette contains tools to create *Entity* elements, *Attribute* within entities, and re-

[3] `https://www.obeodesigner.com/en/product/sirius`



Name: Example
Imported Namespace:

Figure 4: Creating a new conceptual entity

lationships between the same entities (*Relation*). So a dedicated palette with specific tools for each type of database is also available (see Fig. 3 showing the TyphonML graphical editor at work).

Each element carries its attributes specified through the appropriate "Properties" view of Eclipse. See the example in Fig. 4, where a new *Entity* is created through the palette, and the *name* is set through the "Properties" view. It is important to remark that the graphical editor prevents problems of inconsistency between the tools present in the palette. In other words, it is not possible to create elements that are not those expected from the language metamodel. Thus, for example, the "*Attribute*" tool in the section of the palette dedicated to the *Entity* can be inserted only within *Entity* elements.

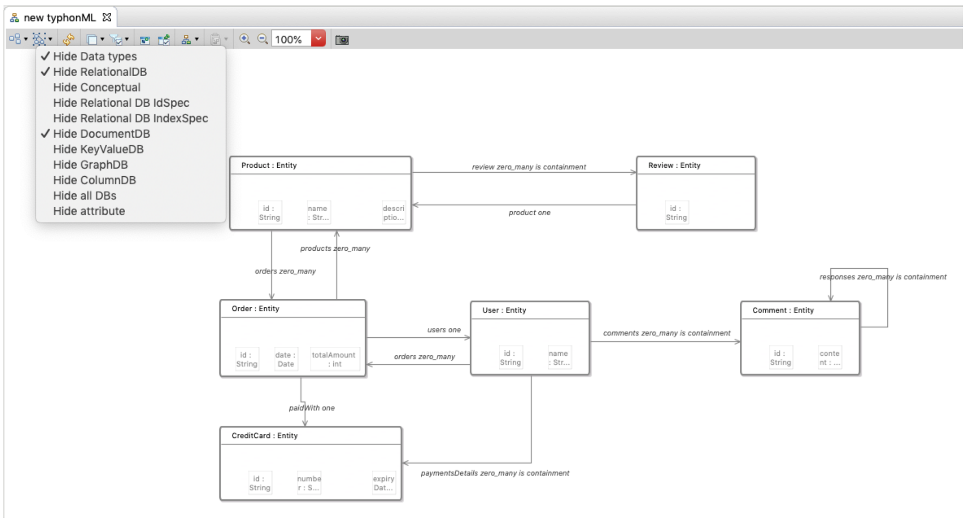


Figure 5: Hiding elements in the graphical presentation of TyphonML models

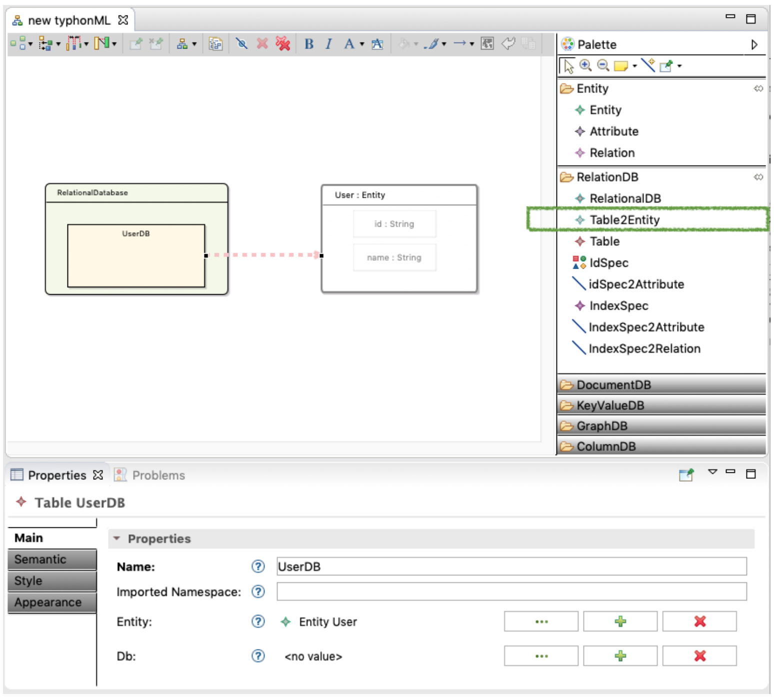


Table UserDB

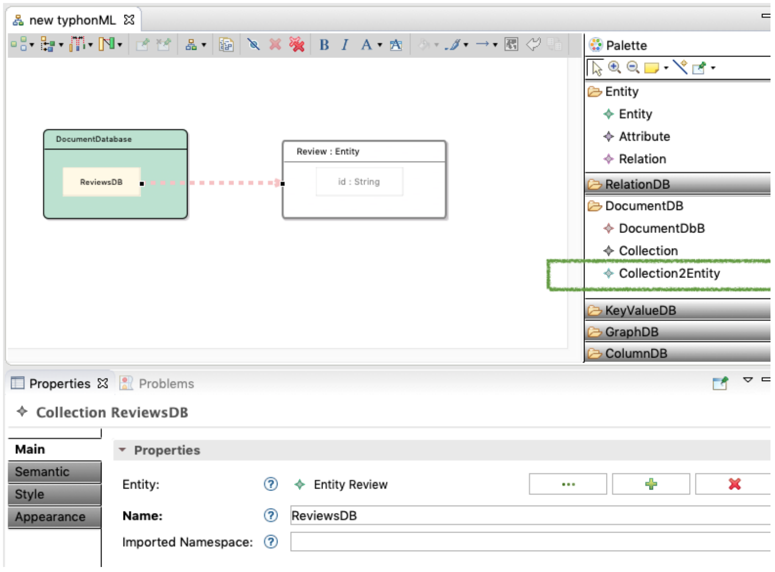
Properties

Name:	UserDB
Imported Namespace:	
Entity:	Entity User
Db:	<no value>

Figure 6: Mapping conceptual entities to relational databases

The editor also permits modelers to hide elements in the canvas to make her easier to view and develop big models. It is possible, for example, to view only the conceptual representation of the model by hiding what concerns the databases and/or vice versa (see an example in Fig. 5).

Mapping conceptual models to relational databases As enabled by the textual language, the TyphonML graphical editor permits modelers to map the conceptual entities to actual database technologies. Figure 6 shows an example of mapping an entity with a relational database. In particular, through the element "*Table2Entity*" of the "*RelationDB*" palette, which is graphically represented by a dashed oriented arrow, the modeler creates a link between a specific relational database table ("*UserDB*") and the selected entity ("*User*").

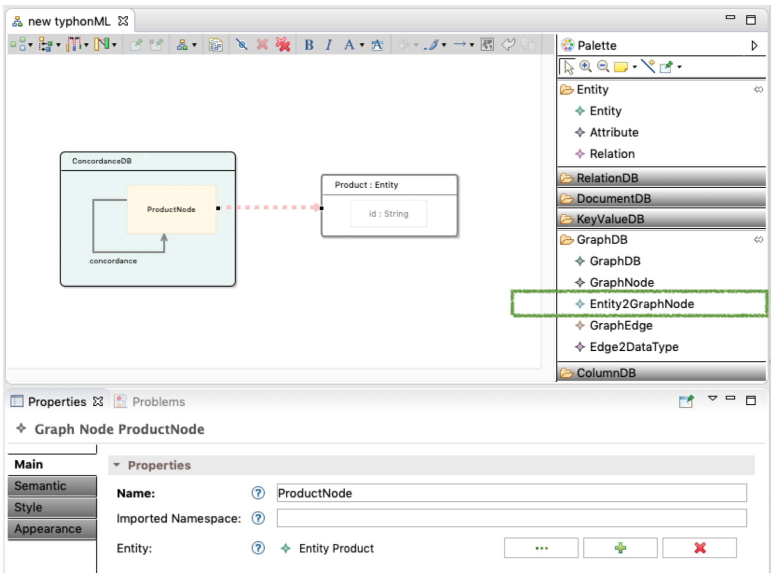


Collection ReviewsDB

Properties

Entity:	Entity Review
Name:	ReviewsDB
Imported Namespace:	

Figure 7: Mapping conceptual entities to document databases



Graph Node ProductNode

Properties

Name: ProductNode

Imported Namespace:

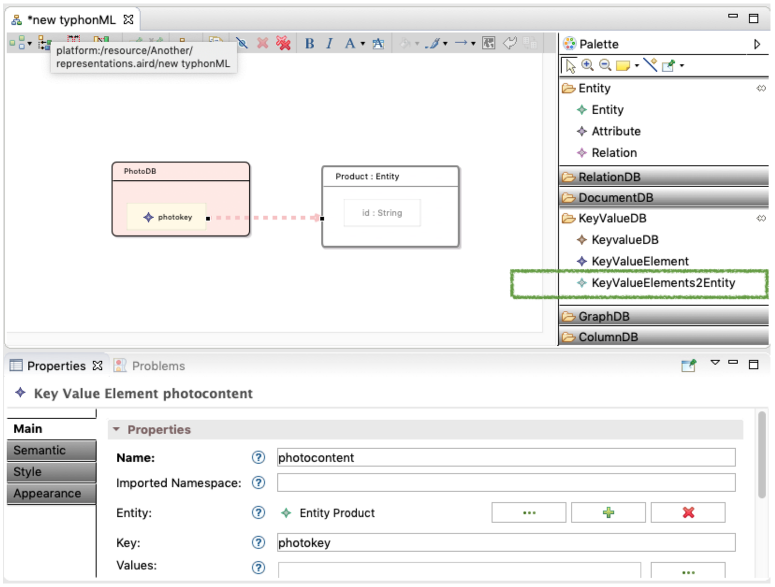
Entity: Entity Product

Figure 8: Mapping conceptual entities to graph databases

Mapping conceptual models to document databases Figure 7 shows how to map a conceptual entity in a document database. In particular, by means of the element "*Collection2Entity*" belonging to the "*DocumentDB*" palette, which is graphically represented by a dashed oriented arrow, the modeler creates a link between a specific document database collection ("*ReviewsDB*") and the selected entity ("*Review*").

Mapping conceptual models to graph databases Figure 8 shows an example of mapping a conceptual entity in a graph database. In particular, by means of the element "*Entity2GraphNode*" of the "*GraphDB*" palette, which is graphically represented by a dashed oriented arrow, the modeler creates a link between a specific graph database node ("*ConcordanceDB*") with the selected entity ("*Product*").

Mapping conceptual models to key-value databases Figure 9 shows how to map a conceptual entity in a key-value database. In the shown example, the "*KeyValueElements2Entity*" element of the "*KeyValueDB*" palette is used to create a link between the "*PhotoDB*" key-value database and the selected "*Product*" entity.



Key Value Element photocontent

Properties

Name: photocontent

Imported Namespace:

Entity: Entity Product

Key: photokey

Values:

Figure 9: Mapping conceptual entities to key-value databases

4 Evolving TyphonML models

Consistently with the work being done in WP6, TyphonML provides modelers with constructs that are necessary to specify changes to be operated on existing TyphonML specifications and thus, to consequently trigger the needed data migration procedures. The supported change operators are those shown in Table 1. It is important to remark that this document discusses the available change operators from a syntactical point of view. The semantics of the available operators in terms of the actual actions that are executed to perform the specified changes is detailed in the deliverable D6.2 [3]. The change operators shown in Table 1 are supported by both the textual and graphical editors even though the usage of the former is recommended from a usability point of view. However, it is important to remark that the TyphonML modeling tools as they have been designed can be easily "fine-tuned" to improve or even extend the support for the needed change operators.

TyphonML element	Level	Change operator
Entity	Conceptual	Add
		Remove
		Rename
		Split
		Migrate
		Merge
Relationship	Conceptual	Add
		Remove
		Rename
		Enable containment
		Disable containment
		Enable bidirectional relationship
		Disable bidirectional relationship
		Change cardinality
Attribute	Conceptual	Add
		Change type
		Remove
		Rename
Table	Logical/Database	Rename
Index	Logical/Database	Add
		Remove
		Add component
		Remove component
Collection	Logical/Database	Rename
		Add Index
		Drop Index

Table 1: Supported change operators (from D6.2 [3])

4.1 Change operators for conceptual elements

Listing 7 shows an example of change specifications involving conceptual elements. The available operators permit modelers to add new entities (line 2), remove existing ones (line 10), and rename them (line 11). Modelers have also the possibility to split an existing entity (line 12) or merge two of them in a new one (line 20). TyphonML permits also to move a conceptual entity from a given database type to another one (line 21).

```
1 changeOperators [
2         add entity newEntityName {
3                 attributes {
4                         att1 : String
5                 }
6                 relations {
 7                         ref1 -> User
 8                 }
 9         }
10         remove entity MyEntity
11         rename entity Product as Item
12         split entity Order {
13                 left entity LeftOrderEntityName {
14                         //...
15                 }
16                 right entity RightOrderEntityName {
17                         //...
18                 }
19         }
20         merge entities Review Comment as NewName
21         migrate Comment to RelationalDatabase
22 
23         add relation newRelationName to Order -> Product [0..*]
24         remove relation "Review.product"
25         rename relation "Product.orders" as newOrdersName
26         change cardinality "User.comments" as 1..*
27         change containment "User.comments" as false
28 
29         add attribute newAttributeName:String to User
30         remove attribute "Product.description"
31         rename attribute "CreditCard.number" as num
32         change attribute "Order.totalAmount" type Real
33 
34         rename table "RelationalDatabase.UserDB" as UserT
35         create tableindex "RelationalDatabase.CreditCardT" {
36                 "CreditCard.number"
37         }
38 
39         drop tableindex "RelationalDatabase.OrderT"
40 
41         extends tableindex "RelationalDatabase.UserDB" {
42                 "User.surname"
43         }
44 
45         reduce tableindex "RelationalDatabase.UserDB" {
46                 "User.name"
47         }
48 ]
```

Listing 7: Specification of TyphonML model changes

It is also possible to change existing TyphonML specifications with respect to conceptual relationships. In particular, relationships can be added (see the added relation named `newRelationName` in the entity `Order` and typed `Product` as defined at line 23), removed (line 24), and renamed (line 25). Moreover, it is also possible to change their cardinality (line 26) and specify different containment prescriptions (line 27). Similarly, it is also possible to add, remove, and rename attributes in conceptual entities, and even change their type as shown at lines 29-32 in Listing 7.

4.2 Change operators for logical elements

TyphonML enables also the specification of changes to be operated at logical level. For instance, concerning modifications that modelers might want to operate on tables in case of relational databases, it is possible to rename existing tables (e.g., see line 34 at Listing 7), and also to specify changes on corresponding table indexes. In particular, modelers can add and remove table indexes (see line 35 and 39, respectively), further than changing the attributes that are considered for indexing the contents of the table being modified. In particular, the keywords **`extends tableindex`** and **`reduce tableindex`** are used to add and remove, respectively attributes in the index of the referred table (see lines 41–45 in Listing 7).

Similarly, TyphonML provides modelers with the change operators that can be applied at logical level on database systems based on collections. In particular, the changes that are supported are collection renaming, addition, and removal of collection indexes.

4.3 Evolving TyphonML models by means of the graphical editor

TyphonML model evolutions can be specified also with the graphical editor. By means of a dedicated *evolution mode*, models can be changed only with the prescribed operators. The editing of a model in evolution mode implies that the starting base is a model already created and deployed, so this visualization mode is provided without palettes.

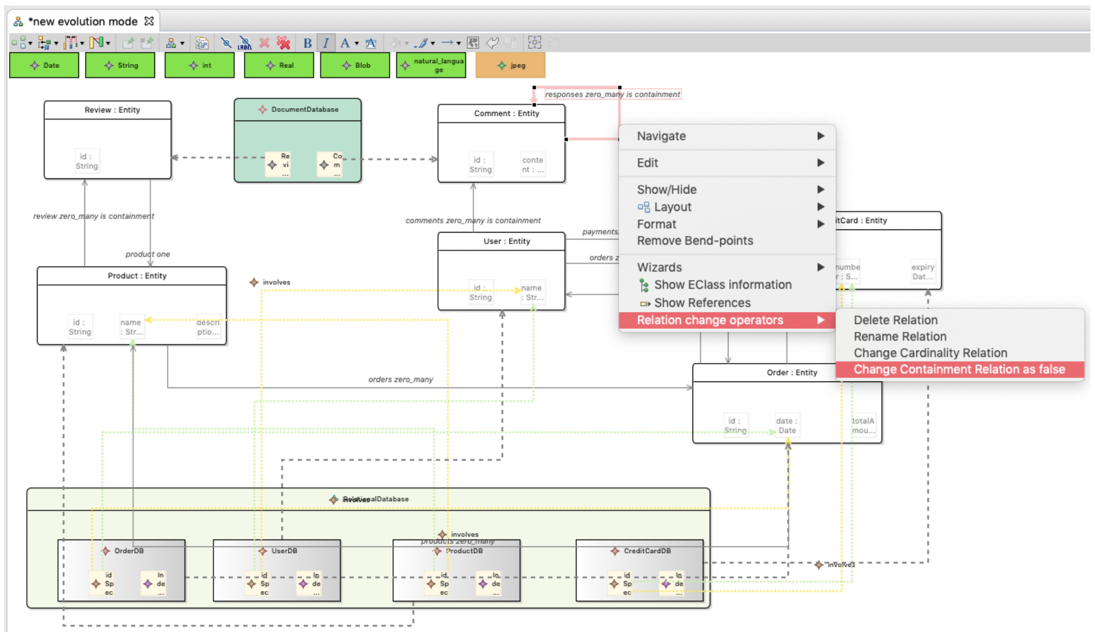
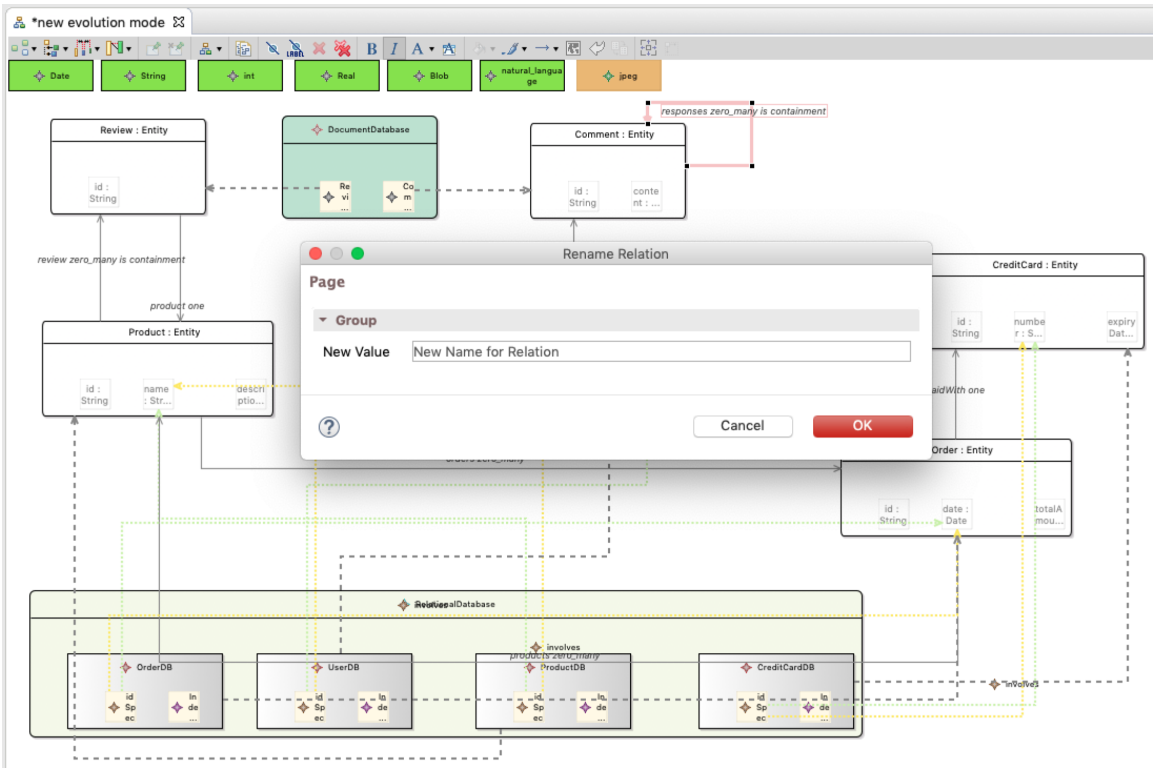


Figure 10: Changing the containment specification of a conceptual relation

By clicking with the right button on an element the system automatically recognizes the type of referring element and based on this it shows an ad-hoc contextual menu with all and only the possible change operators that can be applied on the selected element (e.g., see Fig. 10). A change operator may need additional information as in the case of a renaming in which the new name is requested through a contextual window to enter such information (e.g., see Fig. 11).

As a result, a list of change operator applications is obtained as in the model fragment shown in Fig. 12.



Rename Relation

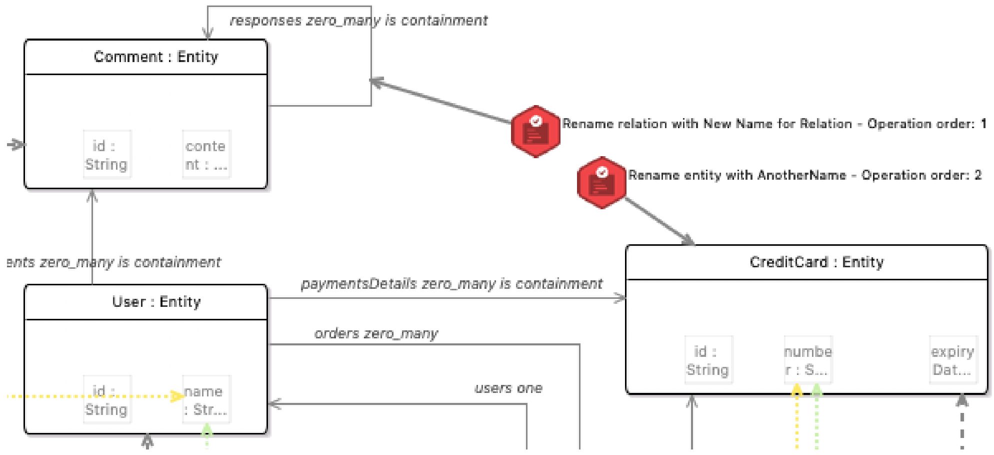
Page

Group

New Value: New Name for Relation

Cancel OK

Figure 11: Renaming a conceptual relation



Rename relation with New Name for Relation - Operation order: 1

Rename entity with AnotherName - Operation order: 2

Figure 12: Ordered application of change operators

5 Validating TyphonML models

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.

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)[4] and Epsilon Validation Language (EVL)[5] 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 non-critical situations that do not invalidate the model, but should nevertheless be addressed by the user to enhance the quality of the model.

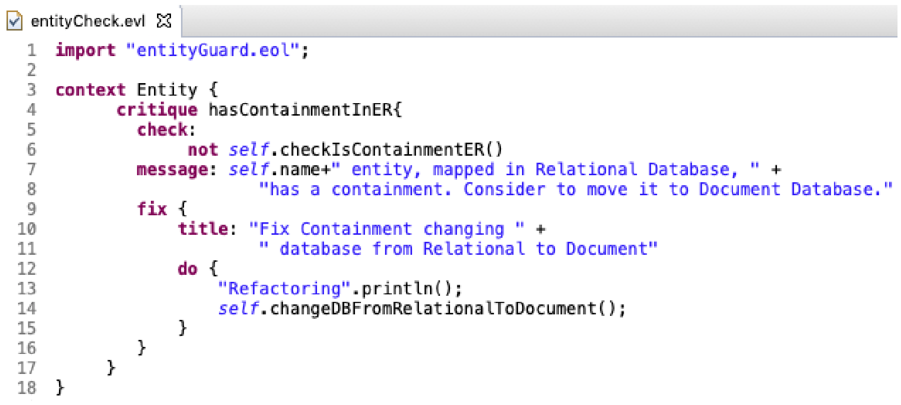
TyphonML modeling supporting tools are endowed with predefined EVL and EOL specifications, even though developers can possibly extend the set of available checks by creating new EVL and EOL files (or extending existing ones). Among the primary checks already available in the corpus there are those related to the `Entity` metaclass. For instance, we are currently able to check if:

> *"One of the specified entity references is a containment and it is also mapped to a relational database."*

Although this is not a real error, the system still provides a warning to the modeler, informing her that probably the best choice would be to avoid mapping this containment to a relational database, and use document databases instead.

Figure 13 shows a fragment of the EVL specification implementing the previously presented check: 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.

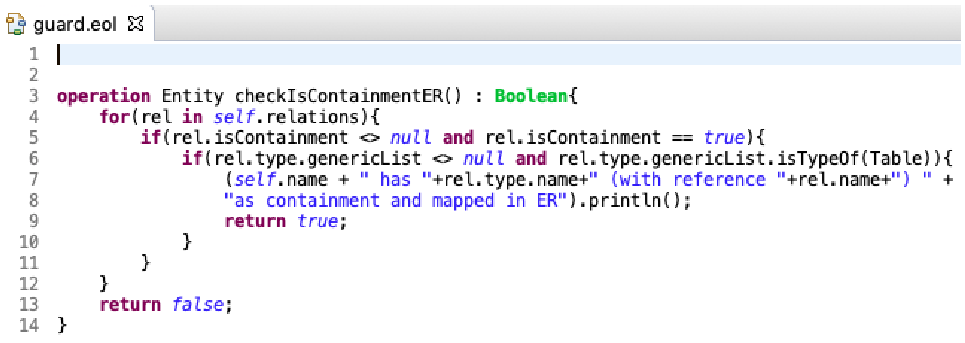
[4] `https://www.eclipse.org/epsilon/doc/eol/`
[5] `https://www.eclipse.org/epsilon/doc/evl/`



entityCheck.evl

```
1  import "entityGuard.eol";
2
3  context Entity {
4          critique hasContainmentInER{
5             check:
6                  not self.checkIsContainmentER()
7             message: self.name+" entity, mapped in Relational Database, " +
8                          "has a containment. Consider to move it to Document Database."
9             fix {
10                title: "Fix Containment changing " +
11                        " database from Relational to Document"
12                do {
13                    "Refactoring".println();
14                    self.changeDBFromRelationalToDocument();
15                }
16            }
17        }
18 }
```

Figure 13: EVL fragment



guard.eol

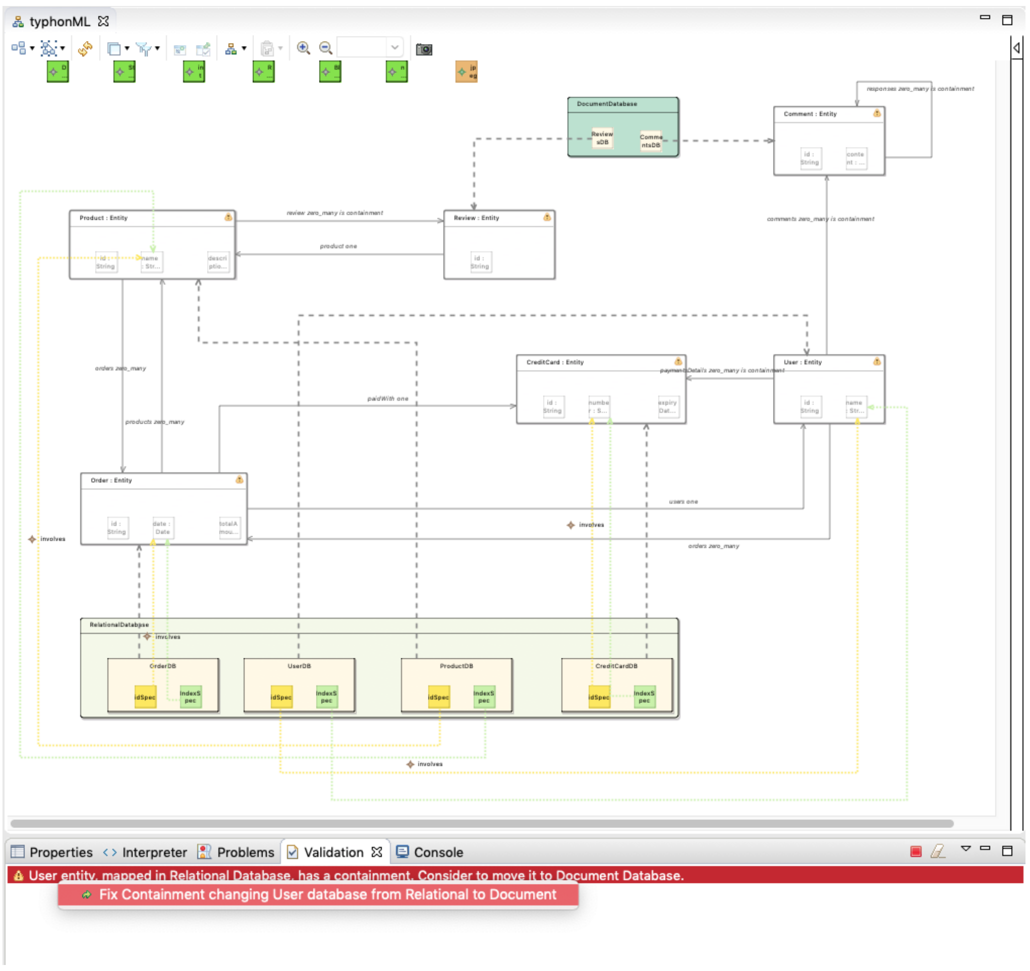
```

operation Entity checkIsContainmentER() : Boolean{
    for(rel in self.relations){
        if(rel.isContainment <> null and rel.isContainment == true){
            if(rel.type.genericList <> null and rel.type.genericList.isTypeOf(Table)){
                (self.name + " has "+rel.type.name+" (with reference "+rel.name+") " +
                "as containment and mapped in ER").println();
                return true;
            }
        }
    }
    return false;
}
```

Figure 14: EOL fragment

The actual check is implemented in EOL as shown in Fig. 14. The defined operation evaluates for each `Entity` instance if among all its references there are some in which the attribute `isContainment` is set to `true` and in that case, it evaluates whether the corresponding entity is mapped to a relational database.

For some checks, it is also possible to provide modelers with automated fixes or suggestions. For instance, Fig. 15 shows a warning related to a matched entity 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. The EOL implementation of the **`operation`** shown in Fig. 16 and called in the fix section of the EVL specification given at line 14 in Fig. 13, first of all it eliminates the *Table* element corresponding to the matched entity and then it creates a new collection in the available document database.

The addition of new checks can be done in two ways: either by inserting a new rule (of the critique type, or constraint) within a context already created (like the one for the entities in Fig. 13), or creating a new EVL file by inserting it in the folder containing all the .evl files.



User entity, mapped in Relational Database, has a containment. Consider to move it to Document Database.

Fix Containment changing User database from Relational to Document

Figure 15: Fix Option from Eclipse Validation View

```
operation Entity changeDBFromRelationalToDocument(){
    for(ent in self.getAllIsContainmentER()){
        var nameOfTheDatabaseToTransfer;
        for(db in self.eContainer().databases){
            if(db.isTypeOf(RelationalDB)){
                for(table in db.tables){
                    if(table.entity == ent){
                        nameOfTheDatabaseToTransfer = table.name;
                        //remove ent from db.tables
                        db.removeTable(table);
                        //add ent to DocumentDB
                        db.addCollectionToDocumentDB(nameOfTheDatabaseToTransfer, ent);
                    }
                }
            }
        }
    }
}
```

Figure 16: Fix example implementation

6 Generating and using the microservice-based TyphonML API

As presented in deliverable D2.3 [2], TyphonML models are used as input to generate a microservice-based API provide developers with the needed functionalities to programmatically develop CRUD operations on the modeled systems. The architecture conceived to manage any polystore is shown in Fig. 17. Specifically, for each modeled database system the corresponding microservice is created, which is responsible of managing all the conceptual entities that have been assigned to that database system.

The architecture consists of a *Client Library* that is a library that can be used by the developers and interacts with the interfaces exposed by the *API Gateway*. The *API Gateway* is the service that knows where all the services managing the conceptual entities are deployed, and thus it is aware of where the client requests have to be forwarded to. The system is also able to manage relationships occurring among entities stored in different database systems, and thus managed by different microservices.

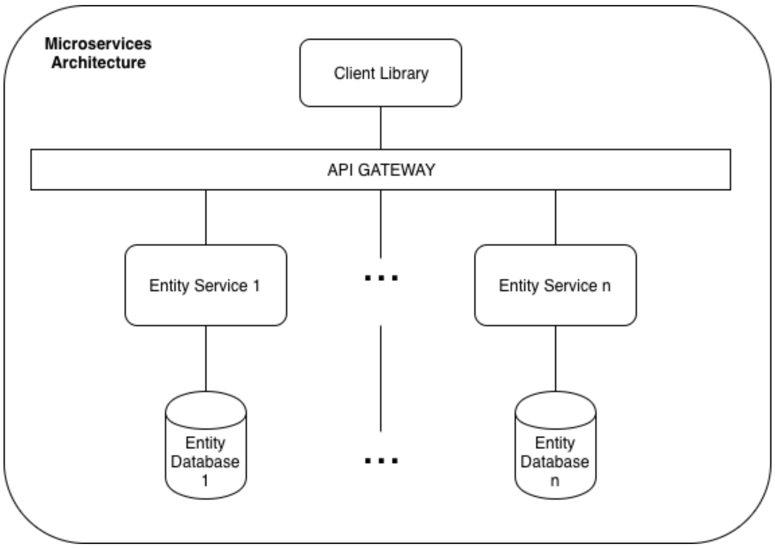


Figure 17: Proposed microservice-based architecture for the Polystore API

The actual implementation of the architecture shown in Fig. 17 relies on the Spring Framework[6]. In particular, Spring Data[7] plays a key role in the proposed approach since it is used to actually access data. Such a layer can be replaced if needed without disrupting the generation of the proposed polystore access infrastructure.

The generation of the data access layer out of an input TyphonML model has been developed as a set of coordinated model-to-code transformations. To this end, Acceleo [8] has been adopted. It is an open-source code project belonging to the Eclipse ecosystem [9] that allows developers to employ model-driven principles to build applications.

Once the *API Gateway*, the microservices for each *Entity*, and the *Client Library* have been created, then it is possible to programmatically use the TyphonML API. Figure 18 shows an explanatory example of the TyphonML API usage. Specifically, a new user is created and given a name through the *user.setName()* invocation; a list of orders is given (*setOrders()*) and it is created by retrieving through the "*orderService*" the

[6]`https://spring.io/`
[7]`https://spring.io/projects/spring-data`
[8]`https://www.eclipse.org/acceleo/`
[9]`https://www.eclipse.org/`

```
public void testCreateUser() {
    User objToCreate = new User();
    objToCreate.setName("Francesco");

    List<Integer> allOrders = new ArrayList<Integer>();
        //Get all the orders in the database
        PagedResources<Order> orders = ordersService.findAll(0,5,"ASC");
        for (Order order : orders.getContent()) {
            allOrders.add(order.getId());
        }

    objToCreate.setOrders(allOrders);
    User userCreated = userService.create(objToCreate);
    System.out.println("User created (id = "+userCreated.getId()+")");
}
```

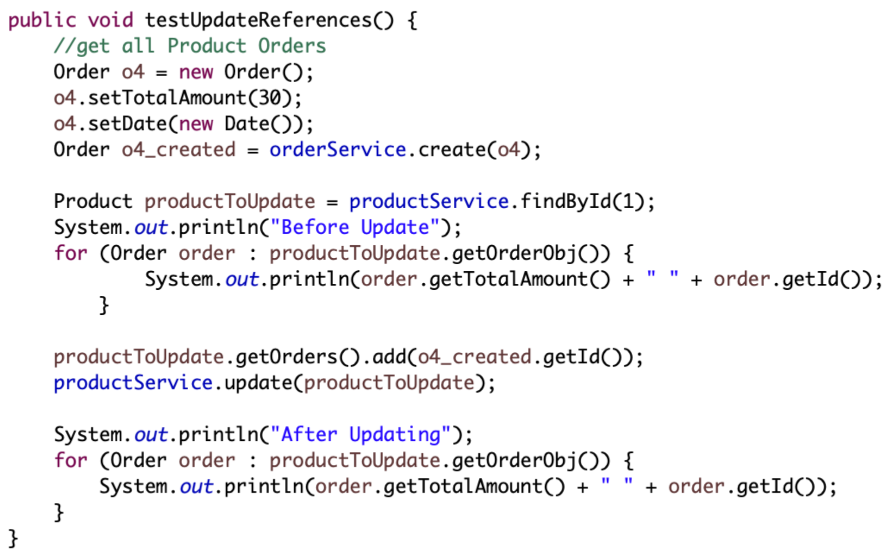
Figure 18: TyphonML API example: User creation with *1xN* relation

first 5 ones in ascending order. In the end, through the service *userService.create()* the user, with the relative relations to retrieved orders (*1xN* relation), is created and saved in the database.

Another example is the one presented in Fig. 19 in which there is another example of creation involving *NxN* relationship. A product (*product*) is created to which different orders are assigned (*product.setOrders()*), and order (*order*) is created to which different products are assigned (*order.setProducts()*). The modification of the

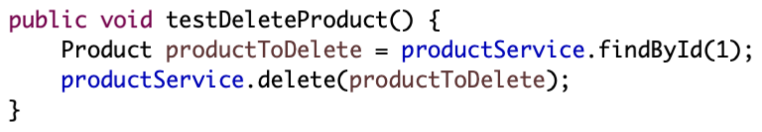
```
public void testCreateProduct() {

    Product product = new Product();
    product.setDescription("A description.");

    List<Integer> orders = new ArrayList<Integer>();
    Order o1 = new Order();
    o1.setTotalAmount(30);
    o1.setDate(new Date());
    Order o1_created = orderService.create(o1);
    System.out.println(o1_created.getId());
    orders.add(o1_created.getId());
    Order o2 = new Order();
    o2.setTotalAmount(60);
    o2.setDate(new Date());
    Order o2_created = orderService.create(o2);
    System.out.println(o2_created.getId());
    orders.add(o2_created.getId());
    Order o3 = new Order();
    o3.setTotalAmount(90);
    o3.setDate(new Date());
    Order o3_created = orderService.create(o3);
    System.out.println(o3_created.getId());
    orders.add(o3_created.getId());

    product.setOrders(orders);

    Product createdProduct = productService.create(product);
    System.out.println("Product created! (id = " + createdProduct.getId() + ")");

    Product anotherProduct = new Product();
    anotherProduct.setDescription("This is another product!");
    List<Integer> anotherOrderList = new ArrayList<Integer>();
    anotherOrderList.add(o2.getId()); // only one order!
    anotherProduct.setOrders(anotherOrderList);
    Product anotherProductCreated = productService.create(anotherProduct);
    System.out.println("Another Product created. (id = " + anotherProductCreated.getId() + ")");

    List<Integer> orderProducts = new ArrayList<Integer>();
    orderProducts.add(createdProduct.getId());
    orderProducts.add(anotherProductCreated.getId());

    Order order = new Order();
    order.setTotalAmount(30);
    order.setDate(new Date());
    order.setProducts(orderProducts);
    Order o4_created = orderService.create(order);
    System.out.println("Created Order with Products (id = " + o4_created.getId() + ")");

}
```

Figure 19: TyphonML API example: Product creation with *NxN* relation



```
public void testUpdateReferences() {
    //get all Product Orders
    Order o4 = new Order();
    o4.setTotalAmount(30);
    o4.setDate(new Date());
    Order o4_created = orderService.create(o4);

    Product productToUpdate = productService.findById(1);
    System.out.println("Before Update");
    for (Order order : productToUpdate.getOrderObj()) {
            System.out.println(order.getTotalAmount() + " " + order.getId());
        }

    productToUpdate.getOrders().add(o4_created.getId());
    productService.update(productToUpdate);

    System.out.println("After Updating");
    for (Order order : productToUpdate.getOrderObj()) {
        System.out.println(order.getTotalAmount() + " " + order.getId());
    }
}
```

Figure 20: TyphonML API example: Product update



```
public void testDeleteProduct() {
    Product productToDelete = productService.findById(1);
    productService.delete(productToDelete);
}
```

Figure 21: TyphonML API example: Product delete

elements is also managed (see Fig. 20). In the example, through the *productService.update(productToUpdate)* method the modification of the element passed as input is made. Finally, deletion is also managed. In Fig. 21 the simple deletion is highlighted through the *delete* method (*productService.delete(productToDelete)*) which cascades eliminate the selected element with all the relative references.

7 Conclusions

In this document we presented the TyphonML modeling tools that have been developed in the context of WP2 and by interacting with the other WPs and partners of the projects including the use case providers, which have given the requirements of the expected technological offerings. The techniques and tools developed in WP2 have been defined to address the requirements that are reported in Table 2.

Req. ID	**Req. description**	**Addressing deliverable**
4	The polystore modelling language shall support storing recorded trend displays	D2.3
6	The polystore modelling language shall support field types and operations to handle spatial data and perform basic operations for ingestion, querying and filtering	D2.3
7	The polystore modelling language shall support a field type that allows to store a Latitude and Longitude values (e.g. "LatLon")	D2.3
8	The polystore modelling language shall support a field type that allows to store a non-geodetic, general X and Y coordinates (e.g. SpatialType)	D2.3
9	The polystore modelling language shall support a field type that allows to store a bounding box (e.g. "BoundingBox")	D2.3
10	The polystore modelling language should provide data types to store binary field	D2.3
11	The polystore modelling language should allow to handle records with binary fields of up to 2 GB	D2.3
12	The polystore modelling language should support additional spatial types as defined by the Lucene interface	D2.3
13	The polystore modelling language should implement the spatial data types and related operations as defined by "OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option"	D2.3
D2	TyphonML shall enable the specification of data models by means of both textual and graphical syntaxes	D2.4
D3	Facilities for generating CRUD APIs from data models specified in TyphonML shall be provided	D2.3
D4	Definition of custom data types to be used in TyphonML data models shall be supported	D2.3
D5	Specification of data types that are needed for applying text-specific analysis (e.g. text, video, recordings) shall be supporte	D2.3
D6	The definition of structured data types (e.g. sentences, facts, entities, events) that can be extracted from text and represented in TyphonML shall be supported	D2.2
D7	The specification of non-functional requirements that will instruct the deployment and querying of the modelled data models shall be supported	D2.5
D8	TyphonML supporting tools shall detect inconsistent data models(e.g. data entities in relational databases that refer to inexistent collections in document-based data models	D2.4, D2.5
D9	TyphonML supporting tools may provide modellers with early feedback about the specified data models (i.e. deployment feasibility of the modelled data with respect to the actual resource availabilities)	D2.4, D2.5
D10	TyphonML editors should be instructed to resolve inconsistencies in the data schema that might be due to system and data evolutions	D2.4
D11	The data migration tools shall define the list of schema changes that can be automatically managed for coupled evolution goals. Such a catalogue of schema changes will be enforced during TyphonML editing sessions that are devoted only to schema evolution purposes	D2.3

Table 2: Summary of the requirements related to WP2

The requirements in bold are those that were not satisfied yet before the work presented in this document as discussed in the following:

D2 - TyphonML shall enable the specification of data models by means of both textual and graphical syntaxes This requirement is now addressed by the textual and graphical modeling tools presented in Section 2 and Section 3, respectively.

D8 - TyphonML supporting tools shall detect inconsistent data models (e.g. data entities in relational databases that refer to inexistent collections in document-based data models) This requirement is satisfied by the functionalities presented in Section 5 that permit to perform early checks on the TyphonML model being specified. Further work investigating the development of additional checks and reasoning tools will be presented in the deliverable *D2.5 – TyphonML Model Analysis and Reasoning Tools*.

D9 - TyphonML supporting tools may provide modellers with early feedback about the specified data models (i.e. deployment feasibility of the modelled data with respect to the actual resource availabilities) The developed validation infrastructure is extensible and as mentioned for the previous requirement, additional early feedback checks will be presented in D2.5.

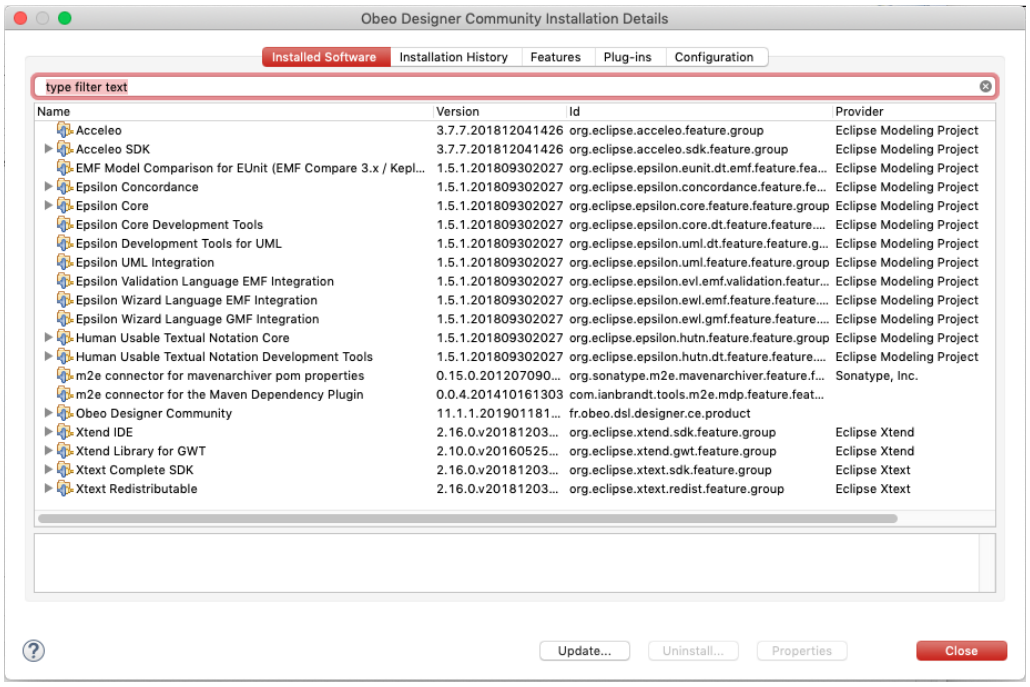
D10 - TyphonML editors should be instructed to resolve inconsistencies in the data schema that might be due to system and data evolutions The fulfilment of such requirement involves a tight collaboration between WP2 and WP6. In particular, the schema and data evolution operators being conceived in WP6 have to be shown to modelers that will trigger them directly from the TyphonML modeling tools. Such an integration has been investigated and implemented as presented in Section 4.

For future work we will continue the collaboration with the TYPHON partners especially in the context of *Task 2.4: Analysis and Reasoning on TyphonML Models*. Moreover, even though this document is supposed to be related to the final version of the TyphonML modeling tools, we will continue their development and support for properly addressing eventual requests for improvement coming from the different tool users.

A Installation and use of the TyphonML modelling tools

To use the previously presented TyphonML modeling tools, the following technical dependencies need to be satisfied as shown in Fig. 22:

- Java 8;
- Sirius: available as Open Source, Sirius is integrated into annual versions of the Eclipse platform[10].
- Acceleo[11]: transformation model-to-model;
- Xtext[12]: an open-source framework for development of programming languages and domain-specific languages;
- Epsilon[13]: a family of languages and tools for code generation, model-to-model transformation, model validation, comparison, migration and refactoring that work out of the box with EMF and other types of models.



Name	Version	Id	Provider
Acceleo	3.7.7.201812041426	org.eclipse.acceleo.feature.group	Eclipse Modeling Project
Acceleo SDK	3.7.7.201812041426	org.eclipse.acceleo.sdk.feature.group	Eclipse Modeling Project
EMF Model Comparison for EUnit (EMF Compare 3.x / Kepl...	1.5.1.201809302027	org.eclipse.epsilon.eunit.dt.emf.feature.fea...	Eclipse Modeling Project
Epsilon Concordance	1.5.1.201809302027	org.eclipse.epsilon.concordance.feature.fe...	Eclipse Modeling Project
Epsilon Core	1.5.1.201809302027	org.eclipse.epsilon.core.feature.feature.group	Eclipse Modeling Project
Epsilon Core Development Tools	1.5.1.201809302027	org.eclipse.epsilon.core.dt.feature.feature....	Eclipse Modeling Project
Epsilon Development Tools for UML	1.5.1.201809302027	org.eclipse.epsilon.uml.dt.feature.feature.g...	Eclipse Modeling Project
Epsilon UML Integration	1.5.1.201809302027	org.eclipse.epsilon.uml.feature.feature.group	Eclipse Modeling Project
Epsilon Validation Language EMF Integration	1.5.1.201809302027	org.eclipse.epsilon.evl.emf.validation.featur...	Eclipse Modeling Project
Epsilon Wizard Language EMF Integration	1.5.1.201809302027	org.eclipse.epsilon.ewl.emf.feature.feature....	Eclipse Modeling Project
Epsilon Wizard Language GMF Integration	1.5.1.201809302027	org.eclipse.epsilon.ewl.gmf.feature.feature....	Eclipse Modeling Project
Human Usable Textual Notation Core	1.5.1.201809302027	org.eclipse.epsilon.hutn.feature.feature.group	Eclipse Modeling Project
Human Usable Textual Notation Development Tools	1.5.1.201809302027	org.eclipse.epsilon.hutn.dt.feature.feature....	Eclipse Modeling Project
m2e connector for mavenarchiver pom properties	0.15.0.201207090...	org.sonatype.m2e.mavenarchiver.feature.f...	Sonatype, Inc.
m2e connector for the Maven Dependency Plugin	0.0.4.201410161303	com.ianbrandt.tools.m2e.mdp.feature.feat...	
Obeo Designer Community	11.1.1.201901181...	fr.obeo.dsl.designer.ce.product	
Xtend IDE	2.16.0.v20181203...	org.eclipse.xtend.sdk.feature.group	Eclipse Xtend
Xtend Library for GWT	2.10.0.v20160525...	org.eclipse.xtend.gwt.feature.group	Eclipse Xtend
Xtext Complete SDK	2.16.0.v20181203...	org.eclipse.xtext.sdk.feature.group	Eclipse Xtext
Xtext Redistributable	2.16.0.v20181203...	org.eclipse.xtext.redist.feature.group	Eclipse Xtext

Figure 22: Sirius Plugins Dependency

Once the TyphonML project[14] is cloned in the local machine, it is necessary to import the project as *Existing Projects into Workspace* (see Fig. 23).

Once all the projects have been imported, we can launch the second instance of Eclipse by right-clicking on a project. (see Fig. 24)

In the new Eclipse instance create a new Modeling Project (see Fig. 25) and give it a name.

[10]`https://www.obeodesigner.com/en/product/sirius`
[11]`https://www.eclipse.org/acceleo/download.html`
[12]`https://www.eclipse.org/Xtext/`
[13]`https://www.eclipse.org/epsilon/`
[14]GitHub repository link: `https://github.com/typhon-project/typhonml`

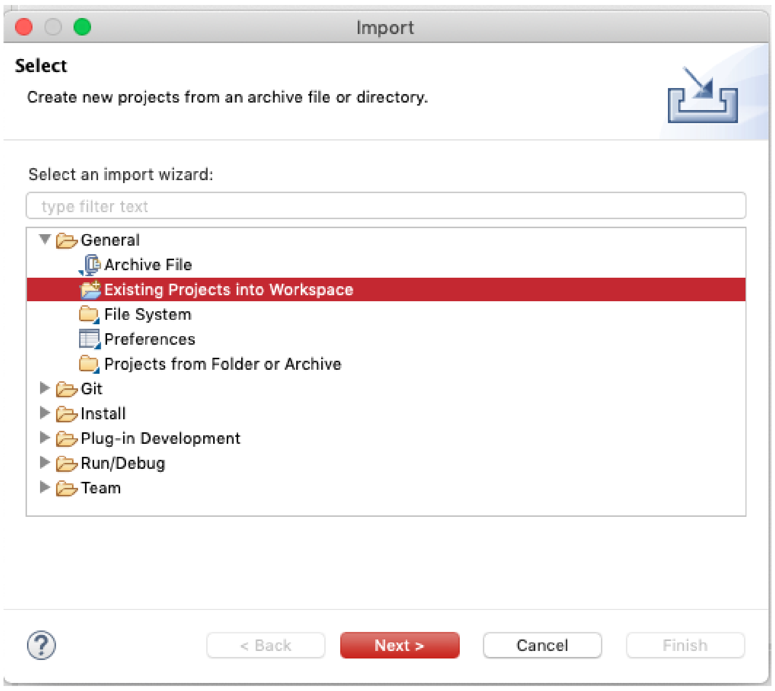


Figure 23: Import Existing Projects

Be sure you have the "Modeling" view open (which is the default) and in that new project automatically you have the *.aird file which contains Sirius representations data (see Fig. 26).

Now you can either create a new model from scratch or import an existing model. Let's assume we want to create one from scratch. We create a file with the .tml extension (it is the extension chosen for TyphonML Xtext files). Convert the project in Xtext Project (See Fig. 27)

Now you can associate the graphical representation to this textual one. Go to the .aird file and as "Representations" select "TyphonML" and create new one selecting, in the next window, the root element "Model" of the model we are creating with the Xtext textual representation. Give to this graphical representation a name.

At this point you can create (both graphically and textually) a new TyphonML model. Each change made and saved on one of the two representations will automatically reflect on the other (See Fig. 29).

Finally, a new entry has been prepared in the Eclipse context menu which allows you to create an .xmi file and another entry that allows you to create all the microservices architecture starting from the .tml file (see Fig. 30).

In Fig. 31 there are the results of the contextual menu.

As far as the evolution mode is concerned, to enable the relative graphical representation, you have to go back to the .aird file and select "evolution mode" (see Fig. 32), create a new one giving it a name. At this point, we will also have the graphical representation of the evolution mode both with the textual representation (first .tml file) and with the new graphical representation.

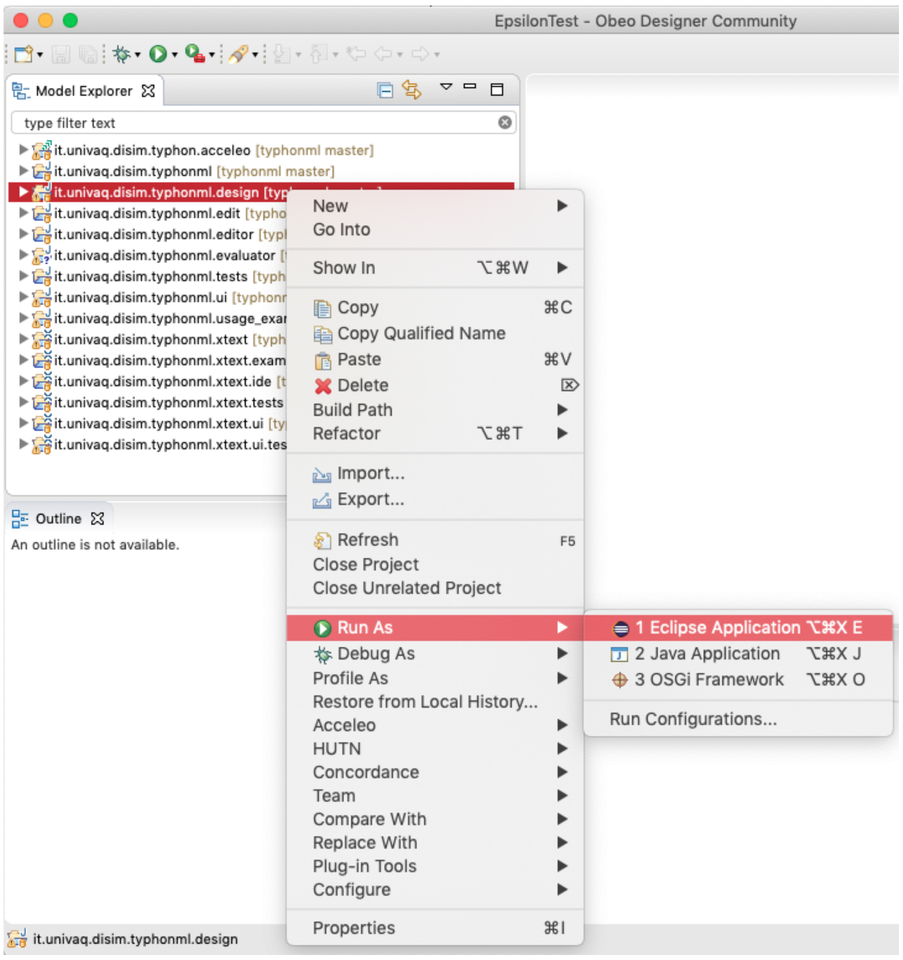

EpsilonTest - Obeo Designer Community
Model Explorer
type filter text
it.univaq.disim.typhon.acceleo [typhonml master]
it.univaq.disim.typhonml [typhonml master]
it.univaq.disim.typhonml.design
it.univaq.disim.typhonml.edit
it.univaq.disim.typhonml.editor
it.univaq.disim.typhonml.evaluator
it.univaq.disim.typhonml.tests
it.univaq.disim.typhonml.ui
it.univaq.disim.typhonml.usage_exan
it.univaq.disim.typhonml.xtext
it.univaq.disim.typhonml.xtext.exam
it.univaq.disim.typhonml.xtext.ide
it.univaq.disim.typhonml.xtext.tests
it.univaq.disim.typhonml.xtext.ui
it.univaq.disim.typhonml.xtext.ui.tes
Outline
An outline is not available.
New
Go Into
Show In ⌥⌘W
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Build Path
Refactor ⌥⌘T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Project
Run As
Debug As
Profile As
Restore from Local History...
Acceleo
HUTN
Concordance
Team
Compare With
Replace With
Plug-in Tools
Configure
Properties ⌘I
1 Eclipse Application ⌥⌘X E
2 Java Application ⌥⌘X J
3 OSGi Framework ⌥⌘X O
Run Configurations...
it.univaq.disim.typhonml.design

Figure 24: Start second Eclipse Instance

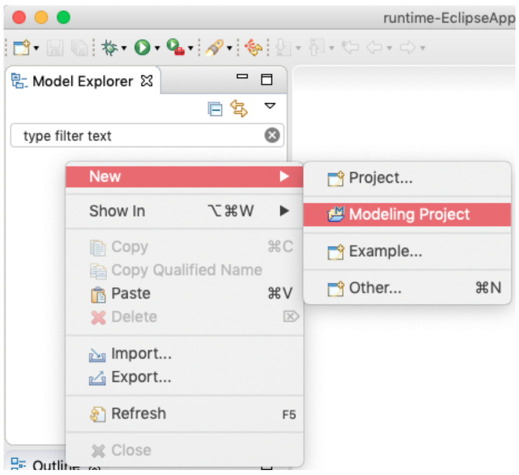


Figure 25: New Modeling Project

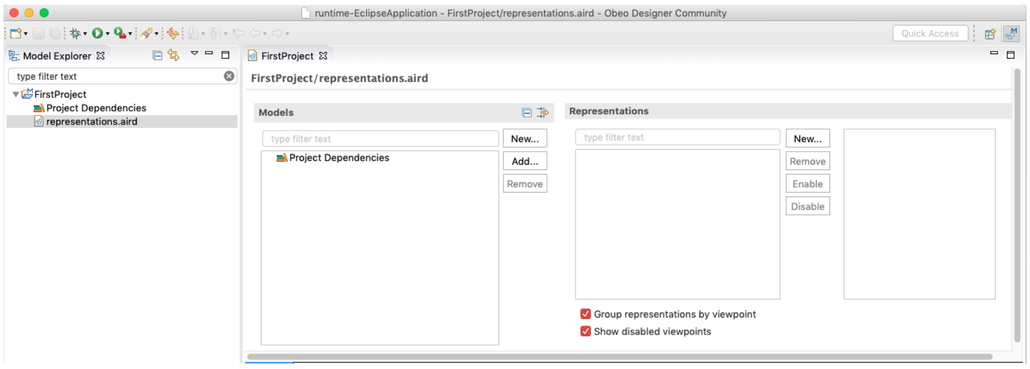


Figure 26: Sirius .aird

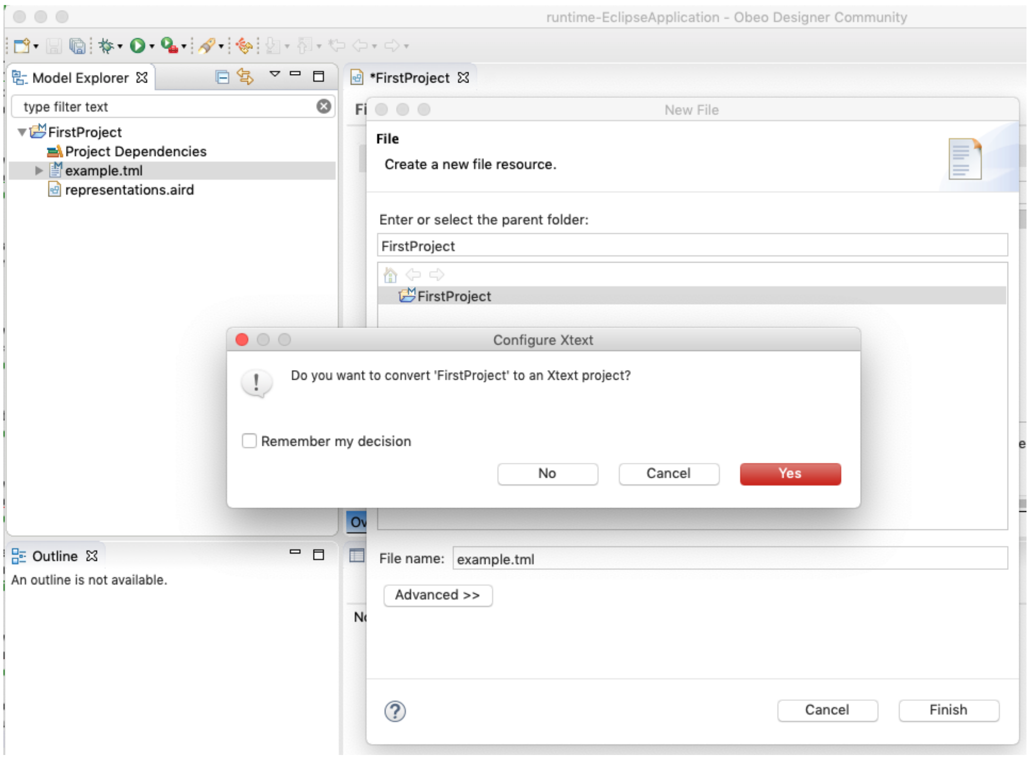


Figure 27: Convert Project to Xtext Project

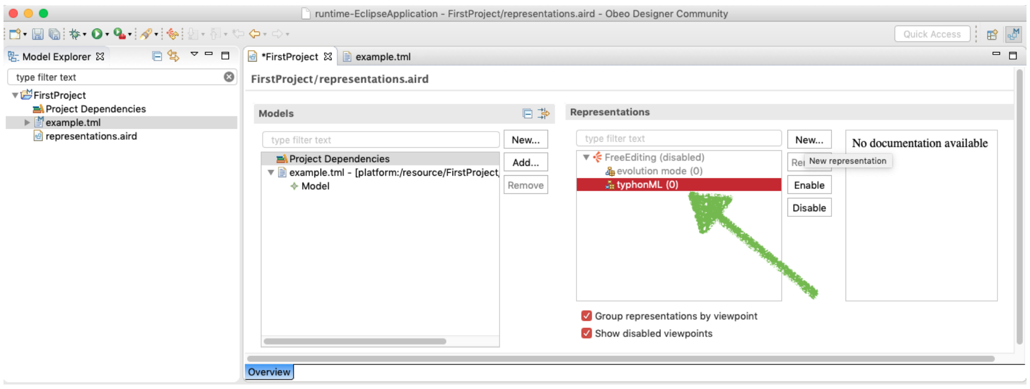
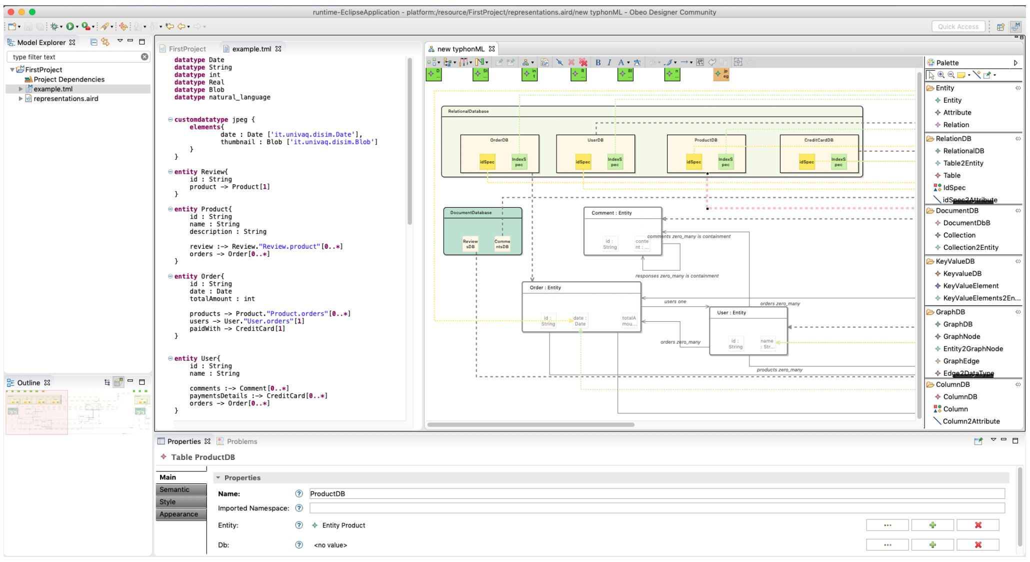


Figure 28: Create new TyphonML graphical representation



```
datatype Date
datatype String
datatype int
datatype Real
datatype Blob
datatype natural_language


customdatatype jpeg {
    elements{
            date : Date ['it.univaq.disim.Date'],
            thumbnail : Blob ['it.univaq.disim.Blob']
    }
}

entity Review{
    id : String
    product -> Product[1]
}

entity Product{
    id : String
    name : String
    description : String

    review :-> Review."Review.product"[0..*]
    orders -> Order[0..*]
}

entity Order{
    id : String
    date : Date
    totalAmount : int

    products -> Product."Product.orders"[0..*]
    users -> User."User.orders"[1]
    paidWith -> CreditCard[1]
}


entity User{
    id : String
    name : String

    comments :-> Comment[0..*]
    paymentsDetails :-> CreditCard[0..*]
    orders -> Order[0..*]
}
```

Table ProductDB

Properties

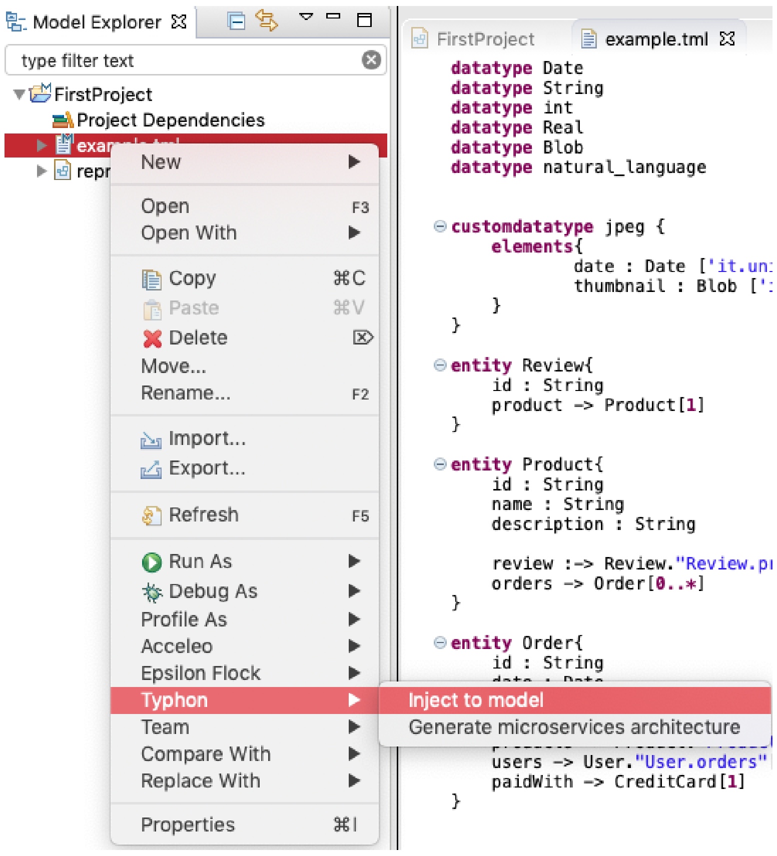
Name: ProductDB

Imported Namespace:

Entity: Entity Product

Db: <no value>

Figure 29: TyphonML Graphical and Textual representation



```
datatype Date
datatype String
datatype int
datatype Real
datatype Blob
datatype natural_language


customdatatype jpeg {
     elements{
               date : Date ['it.un
               thumbnail : Blob ['
     }
}

entity Review{
     id : String
     product -> Product[1]
}

entity Product{
     id : String
     name : String
     description : String

     review :-> Review."Review.p
     orders -> Order[0..*]
}

entity Order{
     id : String

     users -> User."User.orders"
     paidWith -> CreditCard[1]
}
```

Figure 30: TyphonML Contextual Menu

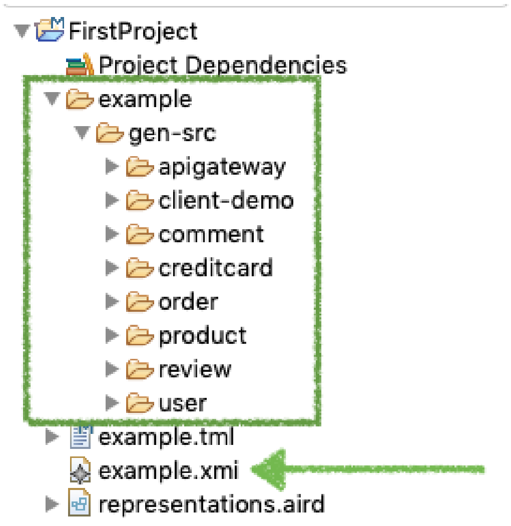


Figure 31: TyphonML Contextual Menu result

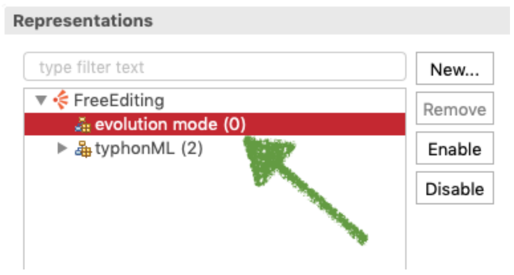


Figure 32: TyphonML Evolution Mode

References

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

[2] The University of L'Aquila. D2.1 Hybrid Polystore Modelling Language (Final Version), 2018.

[3] The University of Namur. D6.2 – Hybrid Polystore Schema Evolution Methodology and Tools, 2018.

[4] The Edge Hill University. D2.2 – Text Modelling Extension, 2018.

[5] The Open Group with contributions from all partners. D1.1 - Project Requirements, 2018.