



**FACULTY  
OF MATHEMATICS  
AND PHYSICS**  
Charles University

**MASTER THESIS**

Bc. Štěpán Stenclák

**Model-driven approach for data schema  
definitions modeling**

Department of Software Engineering

Supervisor of the master thesis: doc. Mgr. Martin Nečaský, Ph.D.

Study programme: Computer Science - Software and  
Data Engineering (N0613A140015)

Study branch: Computer Science - Software and  
Data Engineering

Prague 2022

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....  
Author's signature

Dedication.

Title: Model-driven approach for data schema definitions modeling

Author: Bc. Štěpán Stenclák

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering


Abstract: Abstract.


Keywords: schema definition data modeling ontologies

# Contents


<b>Introduction</b>	<b>2</b>
Ontology . . . . .	3
OFNs and open data . . . . .	3
Focus of the work . . . . .	4
<b>1 Related work</b>	<b>6</b>
1.1 Previous research . . . . .	6
1.1.1 Evolution of schemas . . . . .	7
1.1.2 Implementation . . . . .	7
1.2 Similar work . . . . .	7
<b>2 Analysis and formal description</b>	<b>9</b>
2.1 CIM and PIM layers . . . . .	9
2.1.1 User modifications on the PIM level and evolution from CIM	11
2.1.2 Semantic mapping of related PIM entities . . . . .	13
2.1.3 Evolution of interpreted and non-interpreted PIM entities .	14
2.1.4 Making PIM consistent with CIM . . . . .	15
2.2 PSM layer . . . . .	16
2.3 Whole framework structure . . . . .	17
<b>3 Implementation</b>	<b>18</b>
<b>4 Future work</b>	<b>19</b>
<b>Conclusion</b>	<b>20</b>
<b>Bibliography</b>	<b>21</b>
<b>A Attachments</b>	<b>22</b>
A.1 First Attachment . . . . .	22

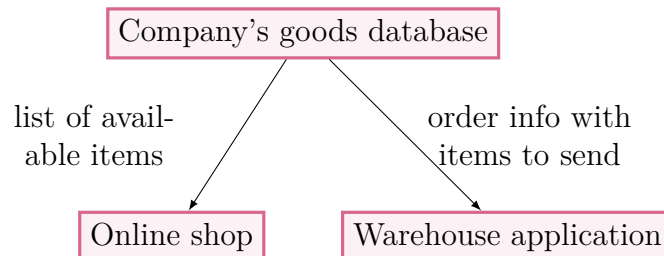
# Introduction


During the software development process, we may come to a point where splitting a large codebase into smaller, well-defined chunks is necessary to maintain the growth of our software. Using sting systems and connecting them is also a viable approach to building software. In both cases, we end up with many applications and services working together.




This approach reduces complexity and demands on software developers as each part can be maintained, deployed, and tested separately. Each developer team must know only the portion they maintain and the immediate surrounding. The surrounding is then defined by a protocol - the face specifying the data which flows between the systems.

Those protocols must be created, documented, and maintained, which can be a long, error-prone task. The result of the process is usually a set of data schemas and documentation for developers. Especially the schemas need to be designed carefully to be, if possible, consistent in format and naming.

As an ample, consider a company selling and distributing its own goods. The goods are stored in warehouses and then shipped to customers. For the shipping process, the warehouse workers need to know the properties of the items they need to send. Similarly, customers need to know the properties of items they are buying. This scenario is denoted in the following image. The nodes are individual systems in the company that communicate with each other.



Our goal is to describe the protocols formally. Which data are sent and where.   
By doing this without any tool, we may find several obstacles during the process:

1. We may need to describe an entity for every schema that uses it. Hence doing the **same work multiple times**. 
2. There is a high chance of **inconsistency**. We may name the semantically identical attributes differently, which may confuse software engineers who came in touch with multiple system parts. 
3. We **lack context**  in the whole system where the entities are used.

Regarding point 1, defining a subschema for shared parts of the domain may not be sufficient, as we may need different detail of information. From the example above, the warehouse application may require only the *name* and *weight* of the items, contrary to the online shop where the *price* is also essential.

In addition to the obstacles, it is hard to provide supporting documentation, diagrams, and examples. 

Future modifications to the system or changes in user requirements may enforce altering the schemas, which again may require changing the same thing multiple times and may cause inconsistency. Formally, we will denote changing of schemas as an **evolution**. The process of evolution is complex, as there can already be existing data that conform to the changed schemas. In that scenario, properly implementing a change in user requirement requires modifying affected schemas, documentation, and all the data (in case the data are stored in the given format).

## Ontology

The example above tends to create a formal description and naming of all entities, their properties, and relations in the given domain. Entities represent objects from the world, such as *order*, *customer*, or *goods*. Relations then specify, for example, that *the order belongs to a customer* and *consists of goods*. Formally, such description is called an **ontology**.

With the data on the web [1] trend in the last few years, even ontologies are becoming accessible publicly. Popular formats include RDFS (or RDF Schema), OWL, UFO, schema.org, and Wikidata. For example, schema.org is a proprietary format describing useful data for search engines, such as events, organizations, or places.

Using pre-defined ontologies in a semi-automatic way of defining schemas is beneficiary because a schema designer may focus entirely on the schema structure and not on the domain semantics.

## OFNs and open data

**Open data** is defined as data published on the web without any restrictions on use. This means that anyone can use, modify and distribute the data for any reason, including commercial use.

The definition of open data is very loose but can be further specified by a 5-star scheme designed by Sir Tim Berners-Lee. Each star adds a restriction up until the fifth star describing the Linked Open Data.

- 1 ★ Data are published on the web and can be used freely.
- 2 ★ Data are structured and in a machine-readable format.
- 3 ★ The format is not proprietary; hence anyone can open them.
- 4 ★ Data uses RDF and SPARQL standards from W3C.
- 5 ★ Data are linked to other data creating a network of data.

According to Act 106/1999 Coll. of the Czech Republic, data of public institutions shall be published as open data on the Internet in all formats it was created, and if possible, in a machine-readable format. This description corresponds to two stars in the schema above.

The act then specifies **OF** (*open formal norms*) as recommendations for publishing selected categories of data, such as information about *Tourist destinations* and *Sports centers*. The purpose of these documents is to standardize

---

<sup>1</sup><https://data.gov.cz/ofn/>

how these data are published, usually by defining JSON and XML schemas along with textual documentation.

Until recently, all those recommendations (OFNs) were created by hand without any tool.

The process of designing those recommendations is comparable to designing schemas for a software system mentioned above - the designer needs to create schemas and documentation for them with examples and images.

## Focus of the work

This thesis will analyze and extend<sup>2</sup> a newly created tool, Dataspecer [7], and formally define and analyze its internal model, which follows previous research in the XML data modeling area and extends it to support new requirements.

Due to the complexity of the topic, this thesis does not attempt to cover and implement all details, as some of them will be analyzed by future work. The thesis only focuses on the current author's work and formalizes the basics.

Dataspecer is a web tool for effortless creation and management of data specifications, such as XSD, JSON Schema, and CSV Schema, and the creation of supplementary documents. The tool helps the user model schemas by providing relations from the chosen ontology. Schemas are modeled in a graphical interface in the form of the tree, which is then converted to various schemas.

Supplementary documents are automatically generated files from the modeled schemas, such as:

- **documentation** - Human-readable description of entities from the ontology, that are used in the schema as well as description of the schema.
- **data transformations** - Scripts that can convert data from one schema to another, even between different technologies, such as JSON and XML.
- **examples** - Small datasets that can be used instead of the documentation to understand the schema.
- **example application** - Depending on the context of modeled schema, if the schema describes web-accessible data, it is possible to generate a web application that uses those data as a proof that the workflow works.

The tool is already in use for creating new schemas for the public sector of the Czech Republic, including the OFNs.

The rest of the thesis is organized as follows.

- The following chapter 1 analyzes related work and introduces the previous works as the common ground this work will follow - precisely the model-driven approach for data modeling and evolution of XML documents.

---

<sup>2</sup>The author implemented the basis of the tool as his research project. This thesis, therefore, introduces advanced constructs and formally formalizes those already implemented.



- Chapter 2 briefly analyzes new requirements for the software as many requirements were already studied in the related work. The chapter focuses on support for schemas other than XML and the data on the web [1] approach to ontology and the modeled schemas. It also defines the internal model to represent schemas and changes from the related work where it was first introduced. Changes are also analyzed concerning the new requirements.
- The next chapter 3 then briefly describes how this model is integrated into the Dataspecer tool.
- The last chapter 4 introduces and briefly analyzes topics for future work, such as the change propagation (evolution) in schemas.

# 1. Related work

This tool follows previous research that was carried out by the *XML and Web Engineering Research Group* (XRG) at the Faculty of Mathematics and Physics of the Charles University in the years 2012 to 2015.

The following section introduces the research's core concepts that will be analyzed and used in the thesis, followed by the section on similar work in the research area.

## 1.1 Previous research

In 2012 XRG formalized a novel approach<sup>1</sup> to modeling XML schemas [6] for a particular domain ontology by integrating Model-Driven Development (specifically Model-Driven Architecture) techniques to separate a conceptual model describing the domain ontology and a structural model which described the concrete XML schema.

Model-Driven Architecture is a software engineering technique that abstracts development into three viewpoints: CIM as the *Computation-independent Model*, PIM as the *Platform-independent model* and PSM as the *Platform-specific model*.

XRG used only the last two layers of the architecture. PIM represents the domain ontology in UML-like notation<sup>2</sup>, as it is independent of the platform as XML. PSM as a platform-specific then represents the given schema in their own designed grammar, which was translated into a final schema, such as XSD.

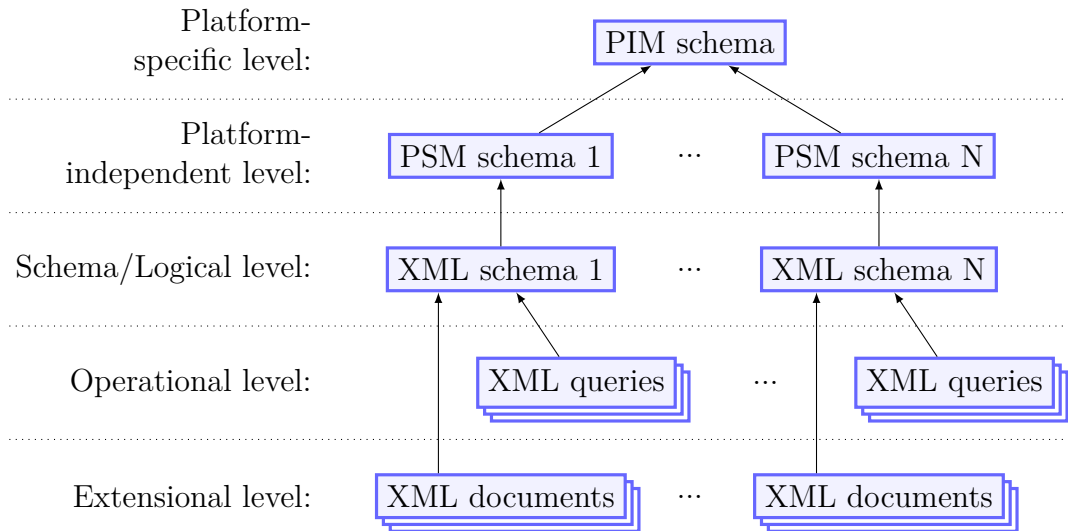


Figure 1.1: The five-level framework proposed by XRG in [6]. A shared PIM layer with a conceptual model is used by multiple PSMs, where schemas are defined in their own grammar. The grammar is then translated into schemas that conform to XML documents.

<sup>1</sup>The work builds on previous work of the same authors where they introduced the XSEM model [4], which has later been a subject of their study.

<sup>2</sup>Formal definition of PIM and PSM levels is in chapter ??.

The major benefit of the strategy presented is a shared conceptual model between various XML schemas, as other works at that time had a conceptual model for every schema. This was not practical as usually multiple schemas are applied in a single software. Authors have also formalized the model and have proven that their approach is correct. That means that (i) every conceptual schema models XML schema, (ii) their translation algorithm from the internal model to schema respects introduced rules and is reversible, and (iii) their normalization and optimization algorithms produce semantically same schema.

### 1.1.1 Evolution of schemas

The focus of XRG was also directed to the evolution [5] of their proposed model to minimize the work of the data designer. As already stated in the introduction of the thesis, changes may be inevitable (either from the user requirements or from the surrounding environment) in large and complex systems, and propagating even a tiny change from the domain ontology to all affected schemas is time-consuming and error-prone.

They proposed, formalized, and later implemented a solution in restricting the changes in PIM and PSM models to only atomic operations - simple changes in the model, such as *creating a new class*, *updating a name of association* or *removing an attribute*. Those operations are not intended to be used by the user directly but are simple enough to be formally defined and mapped to the corresponding operations in the level below. The proposed mapping is then used to propagate changes in the model to the schema level, more precisely from PSM to PIM, which is then translated to the schema level. They implemented only top-down propagation of changes as the propagation from XML documents is usually not meaningful (but theoretically possible).

### 1.1.2 Implementation

Their result were implemented in two tools *XCase* [2] and *eXolutio* [3]. The former one was simpler, focused only on modeling. The latter then supported schema evolution as described in the previous section. Tools let users define the ontology from which the schemas and operations were derived for XML documents.

## 1.2 Similar work

- Google Scholar docela něco vrací na "model driven transformation cim pim data schema".
- Dále pak jsem našel možná relevantní - <https://link.springer.com/article/10.1007/s10270-021-00905-x>. K článkům byste se měl z univerzitní sítě dostat zdarma.

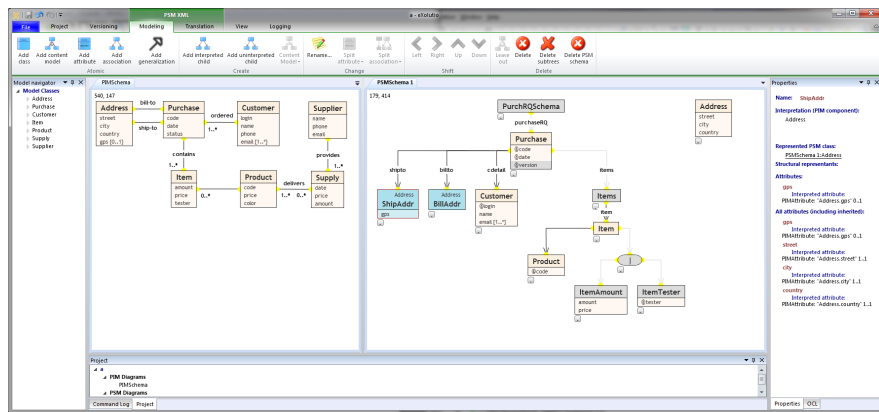


Figure 1.2: Preview of the eXolotio tool. The left panel shows the PIM model as a UML class diagram, while the right panel represents a PSM schema in a tree-like structure that can be converted to an XML schema.

## 2. Analysis and formal description

*This chapter analyzes several requirements regarding framework layers and proposes modifications to them. As will be stated in the chapter 3 Implementation, not everything is implemented yet due to complexity. Nevertheless, it is crucial to properly design and plan everything in advance to minimize the technical debt.*

### 2.1 CIM and PIM layers

**Requirement 1.** As many ontologies are located on the web in formats like OWL, RDFs, UFO, etc., the application shall support reading them. This has the following impacts:

1. The ontology may **not always be available**. Unavailability should not forbid us from generating the schemas and making minor changes to them if those changes are not directly related to exploring the ontology.
2. The ontology may **change unexpectedly**. As there are no strict requirements, we must not expect to get the history of modifications yet still be able to perform schema evolution in some semi-automatic way.
3. The concepts in the ontology may **point to another ontology** according to Linked Open Data principles.

In contrast with the approach introduced in *XCase* and *eXolutio* tools, the process of creating the domain ontology is moved from the application to the external tools. The application then only uses those ontologies if required.

To support the first and second points, we modified the previously introduced five-level framework. We added new most level called CIM (from *Computational Independent Model*). CIM represents the remote ontology on the web. Although there can be more ontologies, with the LOD approach, we can see them as one. PIM is then used as a copy of CIM, and only the necessary entities are copied to it, which corresponds to the previous tools whose ontology was stored in PIM. This modification is in accordance with point 1 and gives us a base ground for implementing point 2, as we can compare CIM and PIM and try to create a sequence of operations that modifies PIM to be consistent with CIM again.

**Requirement 2.** The application shall support multiple data formats and their validation languages (such as JSON Schema, XSD, and CSV schema), and it shall be possible to add support for others easily.

Although requirement 2 primarily affects the PSM level, it also affects the definition of an ontology. As we shall support all kinds of serialization technologies, some of them may not use all the information, which pushes us to define the ontology and PIM level in the most elementary way.

**Definition 1 (PIM).** PIM is a triple  $S = (S_c, S_a, S_r)$  of sets of classes, attributes and associations, respectively such that:

- Attribute  $A \in S_a$  belongs to class  $C \in S_c$ , which is denoted by annotation  $class : S_a \rightarrow S_c$  as  $class(A) = C$ .
- Association  $R \in S_r$  is a set of exactly two association ends  $E_1, E_2$  that are associated with classes similarly as with attributes by  $class : E \rightarrow S_c$ .

We will call classes, attributes, and associations entities. PIM then can be decorated by other various semantic and syntactic **annotations** as it can be seen from the definition above. An annotation is a function on a subset of entity space. We do not require that an annotation must be defined for every entity if not stated otherwise.

- Classes, attributes, associations, and association ends<sup>1</sup> may have title and description, or potentially other describing properties that are not directly used in schema generation. However, the title may be used to propose the naming of entities' labels at the PSM level. (For example  $name(C) = \text{"Tourist destination"@en}$ )
- Classes have a set of other classes that they extend by annotation  $extends : S_c \rightarrow \mathcal{P}(S_c)$ .
- Attributes and association ends have cardinalities.
- Attributes have data types.

why PIM has this specific form?

**Definition 2** (interpretation). We say that PIM entity  $I$  is interpreted if and only if annotation  $interpretation(I) \neq \emptyset$ .

**Definition 3** (CIM). CIM  $O$  is an ontology data set for which function **CIM adapter**  $A$  exists, such that  $A(O)$  is a valid PIM, where every entity  $I$  is interpreted, and the interpretation is unique, stable, and consistent.

The definitions tell us that CIM is everything that can be translated to PIM and that process is stable over time. If one entity in CIM is changed, also the resulting PIM should have changed the corresponding entity. If CIM is stored according to LOD format, then the *interpretation* would be the URI of the CIM entity that corresponds to the given PIM entity.

For simplification, in the rest of the thesis, we may omit that CIM *needs to be translated* to PIM and suppose that it is already in PIM-like format.

---

<sup>1</sup>Because two association ends belong to an association, we may annotate the association itself instead of their ends. However, to simplify things, we say that association ends may also have annotations.

### 2.1.1 User modifications on the PIM level and evolution from CIM

**Requirement 3.** The previous approach of creating the ontology directly in the application is not required, but there should be support for *some* modifications.


As stated in requirement 1, the usual scenario for data modeling consists of reading an ontology from the web as a CIM. This section analyzes the requirement 3, whether there exist situations where modifying the ontology in the tool may be more beneficial than modification of the ontology itself.


We can classify the reasons for modifying the ontology as follows:

1. The ontology is wrong and does not describe the domain correctly. - *Then, a correct way would be to fix the ontology.*
2. The ontology describes only a subset of the domain. Either only the core of the domain or the ontology is complete, but only for one domain, whether in another, something may be missing. - *If the desired ontology is strictly a superset of the domain, we can exploit the linked data features to add missing annotations in our own structured data. Then, we would use the new ontology.*
3. The ontology is not granular enough. Some entities can be represented in more detail than they currently are or vice versa. - *We would need to create a copy of affected classes or use an advanced tool if it exists.*

Suppose the example with goods in the delivery company. Although the goods may be identified by EAN (barcode on items), the software team may prefer their own internal identifiers. There can be reasons for not including the identifier in the ontology, as it is too specific for only a software team, for example. This would correspond to the second category from the list above. The missing attribute then may belong to either the original class or the new extended class. The third category may represent the case when we, for example, need to replace an address with a set of more specific attributes such as *street, number, city, country*, etc.

Although in all the scenarios, the preferred way would be to create a new ontology, it can be too cumbersome and time-consuming, especially if the change is too small. Therefore, the possibility of modifying the PIM should be possible.

This is  must be analyzed concurrently with the second point of requirement 1, that the ontology may change. Ideally, the changes in CIM should be reflected in the resulting schemas; therefore we need a mechanism that tells us what has changed. Having PIM strictly as a copy of CIM may help in this process as we can compare the two levels and *somehow* generate a sequence of operations that modifies PIM. The modifications may be as simple as *changing a class title*, or *changing an association cardinality* to the complex ones such as *join two attributes into one* or *move an attribute to another class*. It is essential to have the changes that complex as they carry the information on how the schemas and their data should be modified, not just the final state.

Direct modification of PIM may break this approach because the mechanism that tells us what has changed would also try to revert all the changes made by the user. To be more specific, we are in d only in those entities that have

interpretation - entities that are linked to CIM. All other non-interpreted entities can be just ignored as they were not created from the CIM.

Formally, it may seem that changing an interpreted entity (and still keeping it interpreted) should not be allowed as the changed entity itself does not represent the ontology anymore. As this may be true, there are still some cases when the change is necessary, especially when the CIM does not give us all the information we need (such as missing cardinality or missing description) or there is an obvious error that needs to be fixed.

We will keep the question of what changes are large enough to lose the entity's interpretation open for further discussion.

This observation, together with the fact that we would have to anyway infer the operations from the difference between CIM and PIM, leads us to a simple solution of allowing a user to modify everything and then prompt him/her to apply only the changes that are relevant.

In the context of other requirements and the framework used, this would also mean that:

- PIM entity that is not consistent with CIM and we do not intend to make it consistent can be uninterpreted. This would stop proposing the user to evolve the schemas according to the newest CIM.
- If CIM changes, the semi-automatic update can respect modifications and ask the user for changes that can not be performed automatically.

To make modifications complete, we also need to delete the entities. As we say, the PIM is a subset of CIM, simply deleting the entity would not be enough, as we would not know whether the entity is deleted or just not described. Therefore we introduce a new annotation that marks the entity as deleted. Deletion is a purely cosmetic feature, as, without it, the sufficient approach would be to ignore the entity. It only forbids users to use it in the lower levels.

**Definition 4** (deleted). PIM annotation *deleted* :  $I \rightarrow \{\text{false}, \text{true}\}$ , where  $I$  is a set of interpreted classes, associations, and attributes, denotes that the interpreted entity is deleted and must not be used in the lower levels.

**Definition 5** (consistency). We say that **annotation of interpreted PIM entity is consistent with CIM** if the CIM entity exists and the value is equal to the value in CIM or in the context of the annotation is a superset<sup>2</sup> of the value in CIM. We say that **interpreted PIM entity is consistent with CIM** if the CIM entity exists and all annotations are consistent with CIM.

From the descriptions above, the inconsistency may happen if the CIM changes or when user modifies the PIM. It is easy to detect it as we can compare entities in PIM with those in CIM. To support the second point from requirement 1, we would need to create a set of atomic operations on the PIM level that make the PIM consistent again. These operations can be propagated up to schemas to apply the change from the ontology.

Chtěl bych ještě zmínit alternativní řešení nad kterými jsem přemýšlel a zjistil jsem, že jsou z nějakého důvodu špatná.

---

<sup>2</sup>Consider, for example, *extends*. As we want PIM to be a subset, we also allow *extends* to be a subset of all classes that the given class extends.



### 2.1.2 Semantic mapping of related PIM entities

The third reason for modifying an ontology from the previous subsection may be analyzed from another point of view. As we decide to introduce a new entity/entities that replace others, it may be beneficial to keep the information that the two groups are semantically connected. This would probably tend us to extend PIM with semantic mapping between entities, which can be used for transformations purposes.

As an example, suppose the address again. Address as a whole can be mapped to its parts, such as street, number, etc., and vice versa. The simplest mapping would split the string by commas and new lines, and join them back, respectively.

The mapping may not be used just for new PIM entities, as we have shown, but also between the entities in the ontology or between different ontologies.

The user then can decide whether he/she wants to use the first or the second group of entities, and the transformation script would still be able to transform data between those two representations.

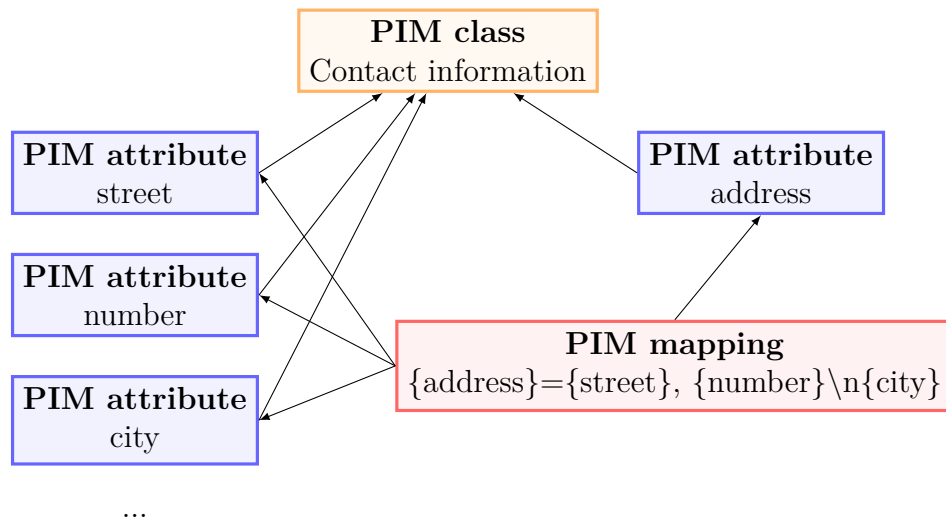


Figure 2.1: Possible solution for the problem of semantic mapping between PIM entities. The example shows two groups of attributes that can be mapped on each other.

This approach, together with data transformations, can even be used to create transformation scripts between old and new data if the ontology changes. We would simply instruct the propagation mechanism not to modify the PIM but rather create new entities and keep the old uninterpreted. We would then copy all schemas that were affected and evolve the copies. This would create new schemas and keeps old ones under the same PIM. Semantic mapping with data transformations would then generate the transformation scripts.

We will keep the question behind the mapping open as it is way too advanced for the current state of the project. However, introducing a new PIM construct to the framework that maps the entities should not collide with the rest of the framework. And as it is at the PIM level, it would be possible to extract this information from CIM.

### 2.1.3 Evolution of interpreted and non-interpreted PIM entities

To demonstrate that the proposed solution may work, we show how some operations may be correctly executed on the PIM level. Suppose we have a sequence of operations that need to be executed on interpreted PIM entities to be consistent with CIM.

Dát sem kódy co se má provést na PIMu, pokud:

- smaže se z CIMu entita. Předtím, než smažu třídu, tak musím smazat zbytek jejich atributů a asociací, ty určitě nejsou interpretovány, protože interpretované byly smazány.
- změni se anotace entity v CIMu. Zeptej se uživatele, jestli ji chce změnit
- vznikne nová entita v CIMu, pak by se hodilo otestovat, jestli už neexistuje lokálně.

#### 2.1.4 Making PIM consistent with CIM

Tady bych navrhl, jak by se mohl PIM zkonzistentnit. Asi by to bylo tak, že se pokusím vyrobit množinu nezávislých změn v PIMu (tady se změnila třída, tady se rozpadl atribut), kdy u každé změny si uživatel vybere, jestli ji provede, nebo ne. Samozřejmě že bych zatím implementoval jen ty jednoduché změny, u ostatních by to napsalo, že neví, co s tím. Otázkou tedy akorát zůstává, jestli tomuto směru aktuálně věnovat energii. (pokud je vůbec správně a nemám v tady nějakou myšlenkovou chybu)

## 2.2 PSM layer

- General definition - most unique
- analysis of basic constructs that may be used, OR and INCLUDE analysis
- News compared to old version - reuse

## 2.3 Whole framework structure

- History of atomic operations is stored with the model because there is a requirement that we may have no access to everything + there is SOLID.
- Everything belongs to schema.
- Because of the first point in this list, the checks need to be performed differently, not during the evolution.
- Operations are stored and can be re-recorded - this allow us to work separately on different parts of the model and then apply the changes by evolution.

### 3. Implementation

- jak je to promitnute v tom toolu, nemusí tam být všechno

## 4. Future work

# Conclusion



# Bibliography

- [1] Newton Calegari, Caroline Burle, and Bernadette Farias Loscio. Data on the web best practices. W3C recommendation, W3C, January 2017. <https://www.w3.org/TR/2017/REC-dwbp-20170131/>.
- [2] Jakub Klímek, Lukáš Kopenec, Pavel Loupal, and Jakub Malý. XCase - A Tool for Conceptual XML Data Modeling. In *Advances in Databases and Information Systems, Associated Workshops and Doctoral Consortium of the 13th East European Conference, ADBIS 2009, Riga, Latvia, September 7-10, 2009. Revised Selected Papers*, volume 5968 of *LNCS*, pages 96–103. Springer, 2009.
- [3] Jakub Klímek, Jakub Malý, Martin Nečaský, and Irena Holubová. eXolutio: Methodology for Design and Evolution of XML Schemas Using Conceptual Modeling. *Informatica*, 26(3):453–472, 2015.
- [4] Martin Necasky. Xsem: a conceptual model for xml. 2007.
- [5] Martin Nečaský, Jakub Klímek, Jakub Malý, and Irena Mlýnková. Evolution and change management of xml-based systems. *Journal of Systems and Software*, 85(3):683–707, 2012.
- [6] Martin Nečaský, Irena Mlýnková, Jakub Klímek, and Jakub Malý. When conceptual model meets grammar: A dual approach to XML data modeling. *Data & Knowledge Engineering*, 72:1–30, 2012.
- [7] Štěpán Stenclák, Martin Nečaský, Petr Škoda, and Jakub Klímek. Dataspecer: A model-driven approach to managing data specifications. In *European Semantic Web Conference*. Springer, 2022.

# A. Attachments

## A.1 First Attachment