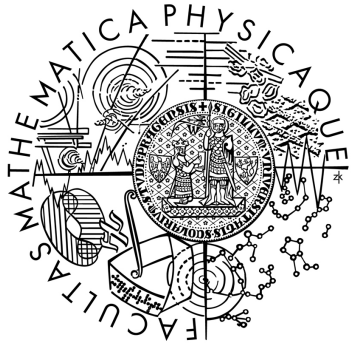
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Eva Jílková

Adaptive Similarity of XML Data

Department of Software Engineering

Supervisor of the master thesis: RNDr. Irena Holubová, Ph.D.
Study programme: Informatika
Specialization: Softwarové systémy

Prague 2013

I would like to thank Irena Holubová for supervising this work and everyone participating in the eXolutio project for providing a framework for it, namely Martin Nečaský, Jakub Klímek and Jakub Malý. Further I would like to thank Jakub Stárka, whose work is expanded in this thesis, for clarification of some problems and Matej Vitásek for the review of this thesis. My thanks goes also to my family and work colleagues for their support and understanding which helped me finish this work.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

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

In Prague Eva Jílková

Název práce: Adaptivní podobnost XML dat

Autor: Eva Jílková

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Irena Holubová, Ph.D.

Abstrakt: V předložené práci studujeme aplikaci podobnosti XML schémat v konceptuálním modelování XML schémat. Pokračujeme v předchozích snahách namapovat XML schémata na PIM schéma pomocí rozhodovacího stromu. V této práci je implementována univerzálnější metoda – rozhodovací strom je natrénován z velké množiny vzorků uživatelsky ohodnocených mapovacích rozhodnutí. Je navrženo několik variací trénování, které by mohly mapování vylepšit. Tento přístup je ohodnocen na velkém množství experimentů, které ukazují výhody a nevýhody navržených variací trénování. Práce také obsahuje přehled přístupů k mapování schémat a popis schémat použitých v této práci.

Klíčová slova: rozhodovací strom, podobnost schémat, PIM schéma, PSM schéma, XML Schema

Title: Adaptive Similarity of XML Data

Author: Eva Jílková

Department: Department of Software Engineering

Supervisor: RNDr. Irena Holubová, Ph.D.

Abstract: In the present work we explore application of XML schema mapping in conceptual modeling of XML schemas. We expand upon the previous efforts to map XML schemas to PIM schema via a decision tree. In this thesis more versatile method is implemented – the decision tree is trained from a large set of user-annotated mapping decision samples. Several variations of training that could improve the mapping results are proposed. The approach is evaluated in a wide range of experiments that show the advantages and disadvantages of the proposed variations of training. The work also contains a survey of different approaches to schema mapping and description of schema used in this work.

Keywords: decision tree, schema similarity, PIM schema, PSM schema, XML Schema

Contents

1. Preface

The XML (eXtensible Markup Language) [9] is a self-descriptive plain-text markup language. It is easy-to-use, well-defined, and yet powerful enough. It has become one of the leading formats for data representation and data exchange on the Internet in the recent years. Due to its extensive usage, large amounts of XML data from various sources is available. It is very useful to adapt independently created XML schemas that represent the same reality for common processing. However, such schemas may differ in structure or vary in terminology. This leads us to the problem of *XML schema matching* that maps elements of XML schemas that correspond to each other.

Schema matching is extensively researched and there is a large amount of applications, such as data integration, e-bussiness, schema integration, schema evolution and migration, data warehousing, database design and consolidation, web site creation and management, biochemistry and bioinformatics. In this work we explore application of schema matching in the area of conceptual modeling.

Difficulties of Schema Matching Matching a schema manually is a tedious, error-prone and expensive work. Automatic schema matching brings significant savings of manual labor and resources. But automatic schema matching is a difficult task because of the heterogenity and imprecision of input data, as well as high subjectivity of matching decisions. Sometimes the correct match depends on the information available or understandable only by a domain expert. In semi-automatic schema matching the amount of user intervention is minimized. The user can provide information before mapping – during the learning phase. Then after the creation of a mapping he can accept or refuse suggested mapping decision, which could be later reused for improvement of further matching.

Conceptual modeling In this work we apply the task of schema matching to a specific application of conceptual modeling – *MDA* (Model-Driven Architecture). MDA [51] models the application domain at several levels of abstraction. Independently developed schemas – *PSM* (Platform-Specific Model) schemas are integrated using a common conceptual schema – *PIM* (Platform-Independent Model).

In the optimal case, the creation of a conceptual model is as follows. Firstly, the PIM schema for a given domain is designed and then the various PSM schemas are derived for specific applications. In reality, the PIM schema has to be designed to describe a domain in a situation where various schemas for specific applications already exist. Independent PSM schemas may come from different sources, they may be of various types and they may use different naming conventions.

Schema matching is used as the key step during this integration process. In particular, we match elements from independent PSM schemas against elements in the common PIM schema to establish the respective PSM-to-PIM mapping. Such mappings then open the possibility of propagation of a change in one schema to all related schemas, ensuring consistency in the whole set of schemas.

This work uses a semi-automatic approach to schema matching. In particular we explore the applicability of decision trees for this specific use case. In our case a decision tree is constructed from a large set of training samples for identification of correct mapping among elements in PIM schema and elements in PSM schemas. Various modifications of the training process are proposed in this work and experimentally evaluated on the basic of several common hypotheses related similarity matching and decision trees.

1.1 Motivation

The work in this thesis is motivated by the following example.

We want to create a common interface that enables to plan different types of vacation - for example ski holidays, holidays at a beach, guided tour etc. at one site. It should be able to integrate existing schemas of travel offers from various travel agencies, allow to search and compare requests according to different criteria and minimize travel costs. Users should be allowed to review their experience and rate provided services. It should provide an interface to select:

- the best means of transport,
- accomodation according to several sets of criteria,
- boarding,
- available facilities,
- auxiliary activities.

The interface should be able to integrate different types of data representation, as the information will be obtained from various types of sources, aggregate various types of customers requests and be easily extendible. Travel agencies, activity providers, facilities and hotels provide their offers in the form of an XML document with an XSD schema. It should be possible to add new sources easily. In context of this thesis, the common interface is the PIM schema and schemas from various sources are the PSM schemas. This interface is shown in Figure 1.1. Elements of XSD schemas are first converted to their corresppponding PSM schema representatives. Then, we need to find the interpretation of the elements against the PIM elements. In our work we focus on this particular task using the idea of schema matching.

1.2 Structure of the Thesis

This thesis is structured as follows. In Chapter 2 we define technologies that are used in this work – XML document, XML tree, XML technologies, PIM and PSM schemas. Schema matching and similarity, along with applications of schema matching is defined in Chapter 3. There is also described usage of schema matching in conceptual modeling. In Chapter 4, related work and existing implementations of schema matching are discussed. Schema matching via decision

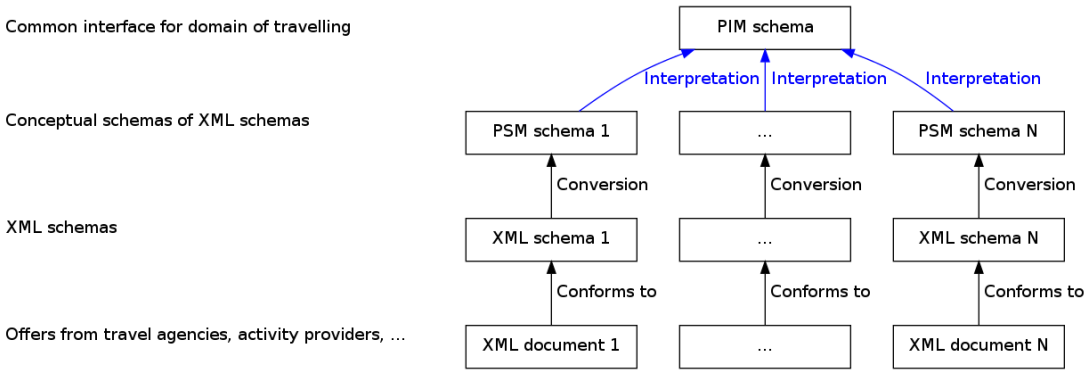


Figure 1.1: Diagram of interface

trees is described in Chapter 5. Chapter 6 describes implementation of our solution. The solution we propose is experimentally evaluated in Chapter 7 and finally results and possible future improvements are briefly resumed in conclusion in Chapter 8.

2. Technologies Used in this Work

This chapter contains a brief description of technologies used in this work – the definition of an XML document, an XML tree and a description of XML schema languages in particular XML Schema, PIM and PSM schema.

2.1 XML Document and XML Tree

An *XML document* is a textual file whose characters are of two types – *markups* that describe the structure of the document, and the actual *content*. It contains the following constructs.

An *Element* can be empty or it contains some subelements.

Example 2.1. An empty element:

```
<book/>
```

Example 2.2. An element with subelements:

```
<book>
  <author>
    Rowling J. K.
  </author>
</book>
```

Element `author` is a subelement of element `book`.

A *Tag* is a mark that encloses elements.

Example 2.3. Start and end tags:

```
<book>
</book>
```

An *Attribute* is a *name*/*value* pair that belongs to an element.

Example 2.4. An attribute:

```
<book title="Harry Potter"/>
```

An *XML Declaration* contains information about the XML document – version of language and character encoding.

Example 2.5. An XML declaration:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

An XML document is *well-formed*

- if it contains an XML declaration,
- if it has a root element,
- if each element is correctly enclosed in a start and end tag or is empty,
- if the start and end tag of each element are written in the same case,
- if the start and end tags of elements are correctly paired - there is no overlap and
- if values of all the attributes are quoted.

An XML document has a hierarchical structure, elements and subelements have a parent-child relationship and can be easily represented as a tree. An example of this representation can be seen in Figure 2.1.

Definition 2.1. An *XML Tree* that represents an XML document D is a labeled tree $T(D)$, where the root of the tree is the root element, internal nodes are element and attribute names and leaf nodes are empty elements, textual content of elements and attribute values. Node a is parent of node b

- if a is a node representing element e_a and b is a node representing subelement e_b of element e_a or
- if a is a node representing element e_a and b is a node representing content of element e_a or
- if a is a node representing element e_a and b is a node representing attribute a_b of element e_a or
- if a is a node representing attribute a_a and b is a node representing value of attribute a_a.

2.2 XML Schema Languages

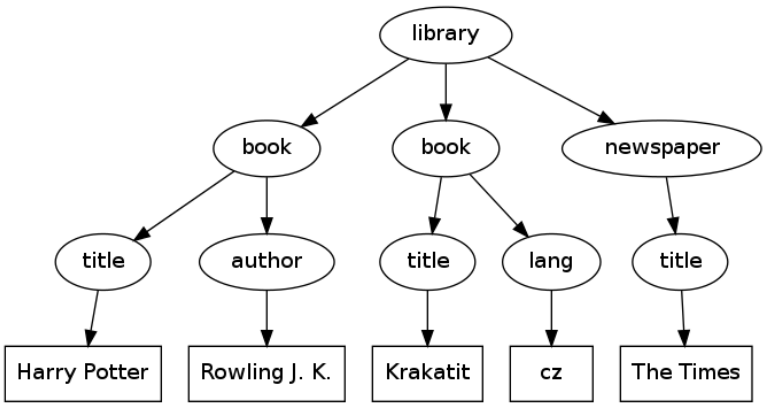
The structure of an XML document is described by XML schema languages such as DTD [9], XML Schema[1] [10], RELAX NG [11] or Schematron [12]. XML document is *valid* if it conforms to the constraints specified in the XML schema.

XML Schema In this thesis, the XML Schema language is used. An example of an XML schema and an XML document that conforms to it is shown in Figure 2.2. XML Schema file is a well-formed XML document file itself, so it has to have the root element and the XML declaration. XML Schema allows to define mainly own data types and the structure of the XML document.

[1]XML schema is the schema of a XML document, whereas XML Schema is one of the languages to describe an XML schema.

```xml
<library>
  <book title="Harry Potter">
    <author>
      Rowling J. K.
    </author>
  </book>
  <book title="Krakatit" lang="cz"/>
  <newspaper title="The Times"/>
</library>
```

(a) XML Document



(b) XML Tree

Figure 2.1: An example of an XML document and its representation as an XML tree

Data types Data type can be simple or complex. Simple data types are derived from built-in types or from the previously defined ones. Built-in data types are e.g. string, integer, boolean, dateTime, or decimal.

Example 2.6. A data type derived from built-in data type string using *restriction*:

```xml
<simpleType name="TypeNotEmptyStr">
  <restriction base="string">
    <minLength value="1"/>
    <maxLength value="20"/>
  </restriction>
</simpleType>
```

Data types can be also derived using *union* and *list*.

Example 2.7. Complex data types are used for definition of element-subelement and element-attribute relations and the amount and order of elements in their parent element.

```xml
<complexType name="TypeAddress">
  <all>
    <element name="street" type="TypeNotEmptyStr" minOccurs="0"/>
    <element name="city" type="TypeNotEmptyStr"/>
```

```xml
<library>
  <book title="Harry Potter">
    <author>
      Rowling J. K.
    </author>
  </book>
  <book title="Krakatit" lang="cz"/>
  <newspaper title="The Times"/>
</library>
```

(a) XML document

```xml
<schema>
  <element name="library">
    <complexType>
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="t:newspaper"/>
        <element ref="t:book"/>
      </choice>
    </complexType>
  </book>
  <element name="book">
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element ref="t:author"/>
      </sequence>
      <attribute name="lang" type="string" use="optional"/>
      <attribute name="title" type="string" use="optional"/>
    </complexType>
  </element>
  <element name="author">
    <complexType mixed="true">
    </complexType>
  </element>
  <element name="newspaper">
    <complexType>
      <attribute name="title" type="string" use="optional">
    </complexType>
  </element>
</schema>
```

(b) XML schema

Figure 2.2: An example of an XML document that conforms to an XML schema

```xml
    <element name="postcode"/>
  </all>
<complexType>
```

`TypeAddress` contains a set of subelements `street`, `city` and `postcode` in no particular order. Subelement `street` is not required.

Example 2.8. Elements in a *sequence* have to be in the particular order:

```
<complexType name="TypePerson">
  <sequence>
    <element name="name" type="TypeNotEmptyStr"
             minOccurs="1" maxOccurs="2"/>
    <element name="surname" type="TypeNotEmptyStr"
             minOcuurs="1" maxOccurs="1"/>
  </sequence>
<complexType>
```

Person could have 1 or 2 names (e.g. Christian name) and only one surname.

Example 2.9. The *choice* element allows to select from the set of elements defined by the choice element.

```
<element name="Price">
    <complexType>
      <choice>
        <element name="FullPrice"/>
        <element name="SalePrice"/>
      </choice>
    </complexType>
  </book>
```

Price could be either full or sale.

Elements An element has a *name* and a *type*. The type can be either defined before, or its definition can be included.

Example 2.10. An element `address` uses a type defined in the previous example.

```
<element name="address" type="TypeAddress"></element>
```

Example 2.11. Element `surname` defines its own type that can not be reused later.

```
<element name="surname">
  <simpleType>
    <restriction base="string">
      <minLength value="2"/>
    </restriction>
  </simpleType>
</element>
```

Attributes An attribute has a *name* and a *type*. Their necessity can be also specified – an attribute can be optional or required.

Example 2.12. An optional attribute:

```
<attribute name="name" type="TypeNotEmptyStr"/>
```

Example 2.13. Attribute `library` is required:

```
<attribute name="library" type="integer"
                          use="required"/>
```

2.3 PIM and PSM Schemas

In this section we define *PIM* (Platform-Independent Model) and *PSM* (Platform-Specific Model) schema. They describe an application domain at different levels of abstraction – a Platform-Independent and a Platform-Specific level, respectively. PIM is a non-hierarchical conceptual schema of a domain that defines the structure and describes the domain independently of the data representation. In general, a *conceptual model* represents entities and relationships among them. Another examples of conceptual models are the E-R model [28] or UML [21]. PSM schemas are manually or automatically derived from the PIM schema and correspond to a specific platform and particular details.

Example 2.14. An example of a PIM and PSM schemas is shown in Figure 2.3. An XML schema is displayed in Figure 2.4. This PIM schema describes the domain of education, where students attend lessons that are taught by teachers. It has the following constructs:

- PIM classes: `Person`, `Teacher`, `Mark`, `ContactInfo`, `Address`, `Lesson`, `Student`.
- PIM attributes: `streetName`, `streetNumber`, `city`, `postalCode`, ...
- PIM associations: `teaches`, `attends`.

PSM schema contains similar constructs:

- PSM classes: `Pupil`, `Address`, `Class`.
- PSM root: `ElementarySchool`.
- PSM attributes: `firstname`, `surname`, `street`, `city`, `name`.

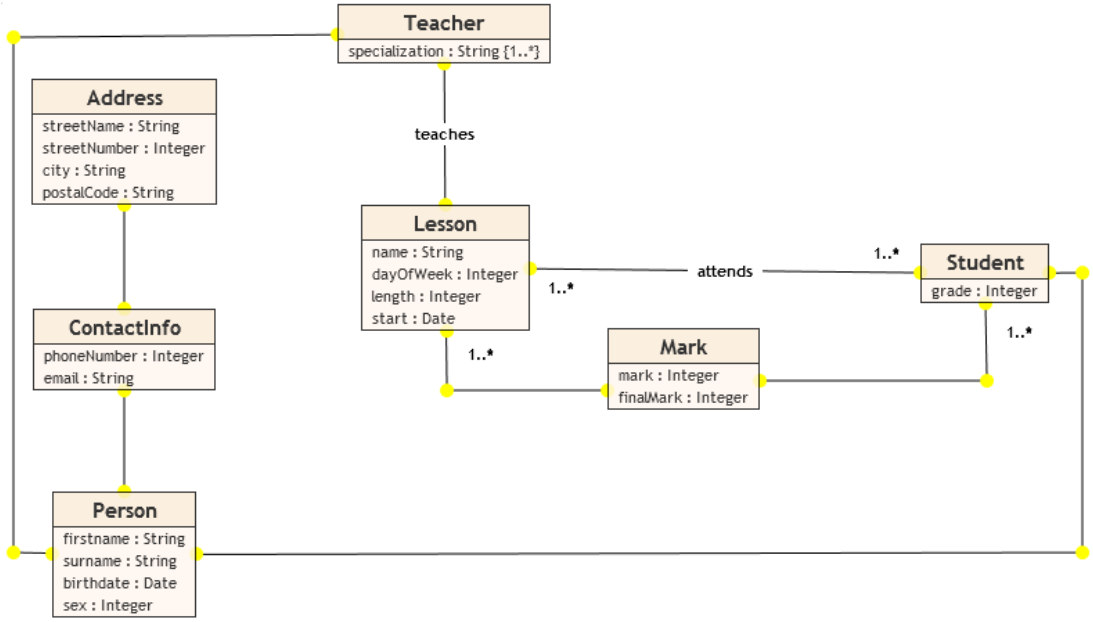
According to [41] the definition of PIM is as follows:

Definition 2.2. A *PIM* is a triple $S = (S_c, S_a, S_r)$ of disjoint sets of *classes*, *attributes*, and *associations*, respectively.

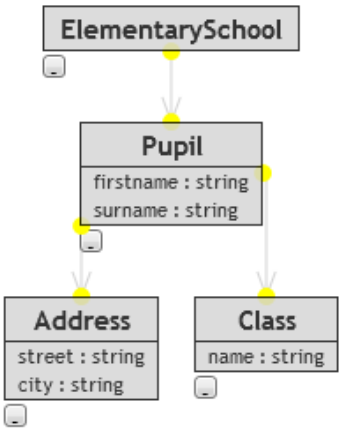
- Class $C \in S_c$ has a name assigned by function *name*.
- Attribute $A \in S_a$ has a name, data type and cardinality assigned by functions *name*, *type*, and *card*, respectively. Moreover, A is associated with a class from S_c by function *class*.
- Association $R \in S_r$ is a set $R = \{E_1, E_2\}$, where E_1 and E_2 are called association ends of R. R has a name assigned by function *name*. Both E_1 and E_2 have a cardinality assigned by function *card* and are associated with a class from S_c by function *participant*. We will call $participant(E_1)$ and $participant(E_2)$ participants of R. $name(R)$ may be undefined, denoted by $name(R) = \lambda$.

PSM schema is defined as follows:

Definition 2.3. A *PSM* schema is a tuple $S' = (S'_c, S'_a, S'_r, S'_m, C'_{S'})$ of disjoint sets of *classes*, *attributes*, *associations*, and *content models*, respectively, and one specific class $C'_{S'} \in S'_c$ called *schema class*.



(a) PIM



(b) PSM

Figure 2.3: An example of PIM and PSM schemas representing the domain of education

```
<xs:element name="elementarySchool">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="pupil">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="adress">
              <xs:complexType>
                <xs:attribute name="streetName" type="xs:string"/>
                <xs:attribute name="city" type="xs:string"/>
              </xs:complexType>
            </xs:element>
            <xs:element name="class" maxOccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="name" type="xs:string"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="firstName" type="xs:string"/>
          <xs:attribute name="surname" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 2.4: An example of XML schema from domain of education

- Class $C' \in S'_c$ has a name assigned by function *name*.
- Attribute $A' \in S'_a$ has a name, data type, cardinality and XML form (whether it models an XML attribute or an XML element) assigned by functions *name*, *type*, *card* and $xform$, respectively. $xform(A') \in \{e, a\}$. Moreover, it is associated with a class from S'_c by function *class* and has a position assigned by function *position* within the all attributes associated with $class(A')$.
- Association $R' \in S'_r$ is a pair $R' = (E'_1, E'_2)$, where E'_1 and E'_2 are called association ends of R'. Both E'_1 and E'_2 have a cardinality assigned by function *card* and each is associated with a class from S'_c or content model from S'_m assigned by function *participant*, respectively. We will call $participant(E'_1)$ and $participant(E'_2)$ *parent* and *child* and will denote them by $parent(R')$ and $child(R')$, respectively. Moreover, R' has a name assigned by function *name* and has a position assigned by function *position* within the all associations with the same $parent(R')$. $name(R')$ may be undefined, denoted by $name(R') = \lambda$.
- Content model $M' \in S'_m$ has a content model type assigned by function *cmtype*. $cmtype(M') \in \{sequence, choice, set\}$.

The graph $(S_c^{'} \cup S_m^{'}, S_r^{'})$ must be a forest of rooted trees with one of its trees rooted in $C_{S'}^{'}$.

There exists mapping from a PSM schema to a PIM schema and is called *interpretation of a PSM schema against the PIM schema*[54]. The mapping specifies the semantics of the PSM schema in terms of the PIM schema and is useful in case of changes in the schema. Change in one schema is propagated to all schemas that are related by the common interpretation, so the set of all schemas remains in a consistent state. In general, an interpretation of a PSM class or attribute is a PIM class or attribute, respectively. An intepretation of a PSM association is not a PIM association directly. It is an ordered PIM association which we call ordered image of the PIM association. It is defined as follows [54]:

Definition 2.4. Let $R = \{E_1, E_2\} \in S_r$ be an association. An *ordered image of* R is an ordered pair $R^{E_1} = (E_1, E_2)$ (or $R^{E_2} = (E_1, E_2)$). We will use $\overrightarrow{S_r^{'}}$ to denote the set of all ordered images of associations of $S^{'}$, i.e. $\overrightarrow{S_r^{'}} = \bigcup_{R \in S_r^{'}} \left\{R^{E_1}, R^{E_2}\right\}$.

An *interpretation of a PSM schema* $S^{'}$ *against a PIM schema* S is defined as follows[54]:

Definition 2.5. An *interpretation of a PSM schema* $S^{'}$ *against a PIM schema* S is a partial function $I : (S_c^{'} \cup S_a^{'} \cup S_r^{'}) \rightarrow (S_c \cup S_a \cup \overrightarrow{S_r^{'}})$ which maps a class, attribute or association from $S^{'}$ to a class, attribute or ordered image of an association from S, respectively. For $X^{'} \in (S_c^{'} \cup S_a^{'} \cup S_r^{'})$, we call $I(X^{'})$ *interpretation* of $X^{'}$. $I(X^{'}) = \lambda$ denotes that I is not defined for $X^{'}$. In that case, we will also say that $X^{'}$ *does not have an interpretation.*

Let a function $classcontext_I^{'} : S_c^{'} \cup S_a^{'} \cup S_r^{'} \cup S_m^{'} \rightarrow S_c^{'}$ return for a given component $X^{'}$ of $S^{'}$ the closest ancestor class to $X^{'}$ so that $I(classcontext_I^{'}(X^{'})) \neq \lambda$. The following conditions must be satisfied:

- $I(C_{S'}^{'} = \lambda)$
- $(\forall C^{'} \in S_c^{'} s.t. repr^{'}(C^{'}) \neq \lambda)(I(C^{'}) = I(repr^{'}(C^{'})))$
- $(\forall A^{'} \in S_a^{'} s.t. I(A^{'}) \neq \lambda)(class(I(A^{'})) = I(classcontext_I^{'}(A^{'})))$
- $(\forall R^{'} \in S_r^{'} s.t. I(child^{'}(R^{'})) = \lambda)(I(R^{'}) = \lambda)$
- $(\forall R^{'} \in S_r^{'} s.t. I(child^{'}(R^{'})) \neq \lambda)$
  $(I(R^{'}) = (I(classcontext_I^{'}(R^{'})), I(child^{'}(R^{'}))))$

The interpretation $I(X^{'})$ of component $X^{'}$ in the PSM schema determines the semantic of $X^{'}$ in terms of the PIM schema.

Figure 2.5 displays interpretation of PSM elements against PIM elements of schemas in Figure 2.3.

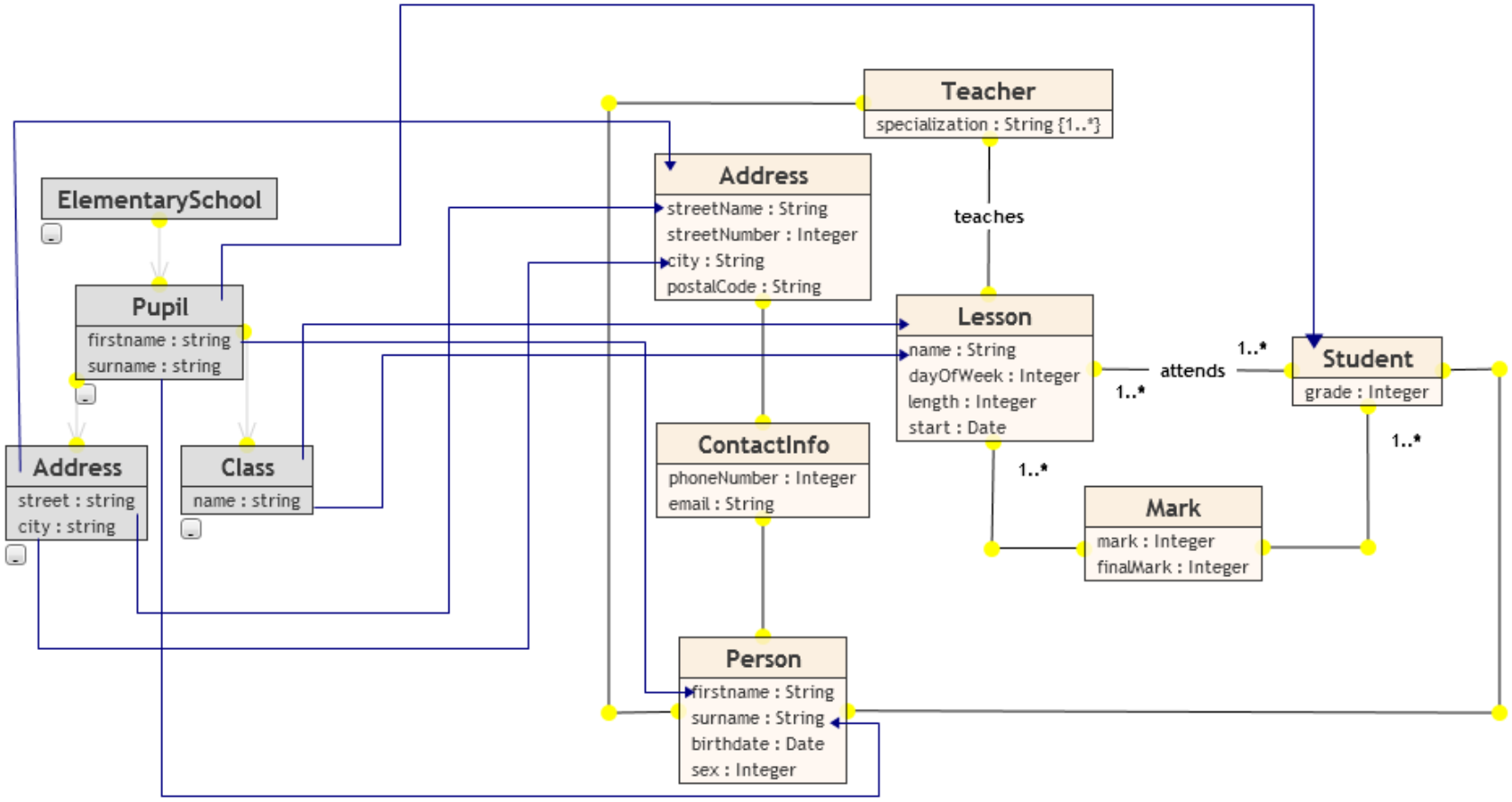


Figure 2.5: Interpration of PSM elements against elements in PIM schema

3. Schema Matching and Similarity

We define basic terms for schema matching in the following paragraph. The semi-automatic or automatic proces of finding correspondences between elements of two schemas is called *schema matching*. In the text of this thesis the term schema matching is used for simplicity, but there are other possibilities.

- *Schema - schema matching* has as an input of two XML schemas.
- *Instance - instance matching* has as an input of two XML documents.
- *Schema - instance matching* has as an input of an XML document and an XML schema.

Similarity is a measure that expresses the level of correspondence. Its value is from interval $[0, 1]$, where 0 means no similarity and 1 means that the compared items are equal in their meaning. *Matcher* is an algorithm that evaluates similarity of schemas according to particular criteria.

3.1 Applications of Schema Matching

Schema matching is extensively researched and has a lot of applications [3].

Data Integration In this particular area there is a set of independently designed schemas and the task is to create a single *mediated schema* that allows a uniform access to it. The independently developed schemas have often different structure and terminology, but they still describe the same real-world model. Schema matching is the first step in data integration proces. It is used e.g. in [39]. *Conceptual Modeling* enables a slight variation of schema integration. Independently developed schemas are integrated using a given conceptual schema.

E-business In bussiness transactions messages with different format are often exchanged and they have to be transformed - we need a conversion between different names, different data types, different ranges of values and different structure. Schema matching is used for integration of different representations of the same concept developed by different parties involved in the bussiness transactions. Examples of this application are [33], [34].

Biochemistry and Bioinformatics Data management in bioinformatics and biochemistry is used for example in genome research, network analysis of molecular interactions, interaction maps of proteins. Information systems contain usually very large data sets. The volume of data grows exponentially as new types of data emerge. In addition, the semantics of biological data is very rich. Schema matching enables to share and reuse data from the previous experiments from heterogenous sources. It is studied, e.g., in [35], [36], [37].

Ontology Matching An *ontology* is a representation of knowledge about a certain domain. It uses *concepts*, *attributes* and *relations* to express this knowledge. The concepts are entities and relations express relationships among them. Ontologies are organized into a taxonomy tree and can be specified by languages such as OIL[1] [49], RDF[2] [50], OWL[3] [52] and SHOE[4] [53]. The ontology matching is problem of finding semantic mapping between the elements of ontologies. Ontology matching is explored e.g. in GLUE [8], GOMMA [31] and LogMap [32].

Data Warehouse A data warehouse is a database with decision support. It is mainly used for reporting and data analysis. Data is extracted from a set of data sources and have to be transformed into the warehouse format. Schema matching is used to find semantic correspondences between elements of source and warehouse schemas. This application is explored e.g. in [38], [46].

3.2 Usage of Schema Matching in MDA

Assume that XML schema in Figure 2.4 was created before the PIM schema in Example 2.14 and we would like to integrate it to the set of PSM schemas. Elements of XSD schemas are converted to their correspponding PSM schema representatives. This conversion is straightforward, as it is proved in [54]. For full integeration we need to find the interpretation of its elements against the PIM elements. This could be done either manually or using schema matching. Manual integration is time-consuming and expensive and that is why we explore usage of schema matching for this task in our work. The PSM element - PIM element pair is identified as an interpretation of PSM element against PIM element if it is suggested as a match by schema matching.

[1] Ontology Inference Layer
[2] Resource Description Framework
[3] Web Ontology Language
[4] Simple HTML Ontology Extensions

4. Related Work

This chapter starts with classifying of schema matchers and continues with surveying several approaches to schema matching. It also describes measures to evaluate the quality of matching decisions suggested to us by matchers.

4.1 Classification of Schema Matchers

In [3], classification of matching algorithms according to several criteria was proposed. It is briefly described in the following text.

Instance vs Schema Matching These two approaches differ in the source of information that are considered as input for matching. Schema matchers use only schema information - *element names*, *descriptions* and *comments*, *relationships*, *constraints* and *data types*, while instance matchers use real instance data.

Instance-based matchers are used on their own (e.g. if no schema is available) or they can improve schema-level matching decisions (e.g. choosing between equally similar elements). Some matching techniques are especially appropriate for instance-based matching, for example *linguistic characterization* of text content of elements (frequencies of words, keywords) and *constraint-based characterization* of numerical attribute values (value ranges and averages) or of textual content of elemenents and textual values of attributes (character patterns). Sample approaches are Tree Edit Distance algorithms [25], [26] and Time Series Comparing [24].

Element vs Structure Matching Another category of matchers is distinguished by the level of granularity. Element-level matching produces mapping between elements of input schemas. It is used e.g. in [27]. Structure matchers consider complex schema structures – combinations of elements. Each item in the structure is required to match for *full structural match* or only some of them for *partial structural match* (comparison of subschemas of different domains). Structure matching is used, e.g., in [48], [4].

Language vs Constraint Matching Language matchers use either *names*, *texts* or *descriptions* for finding semantically equal or similar ones. During linguistic processing, an *auxiliary source of information* is needed – thesaurus or dictionary of synonyms and hypernyms, multi-language dictionary, domain-specific dictionary, list of abbreviations. Examples of language matchers are described in Subsection 4.2.1. Comments associated with schema elements can be used as an additional source of information.

If *constraints* (data types, value ranges, keys, optionality of attributes, cardinality or relationships) of schema elements or attributes are compatible, it suggests a possibility of a match between them. This approach is used, e.g., in SemInt [27].

Combination of Individual Matchers It is neccessary to combine multiple different approaches for an effective matching. A single matcher focuses on a particular feature and could miss some good match candidates. Combinations allow us to use the criteria from different categories – e.g. structure-level matching with name matching. There are two possible ways of combining the individual matchers.

- Use different match criteria or properties within a single *hybrid matcher*.
- Aggregate the results of several independently executed match algorithms in a *composite matcher*.

A hybrid matcher should also achieve better performance than the separate execution of multiple matchers – e.g. poor match candidates are filtered out early, the number of passes over the schema is reduced by computing more criteria at once for an element before continuing to another one. Individual matchers can be evaluated simultaneously or in a specific order. Hybrid combination of individual matchers is used in Similarity Flooding algorithm [5] and CUPID [4] matcher.

A composite matcher integrates results of one-approach matchers or hybrid matchers. It is a more flexible way of combining than the hybrid matcher, as it provides the possibility of reordering the sequence of matcher execution and the possibility of selecting appropriate matchers from a fixed set based on application domain or language of input schemas. Selection of matchers can be done automatically or manually. Composite approach is used in e.g. COMA [1], COMA++ [15], LSD [7], Bayesian Networks [42] matchers.

The proposed criteria are hierarchically ordered and form a classification tree that is displayed in Figure 4.1.

Table 4.1 classifies several approaches according to proposed criteria.

	COMA	SF	DT	Bayesian Networks
Combination type	Composite	Hybrid	Hybrid	Composite
Composition	Manual	-	-	Automatic

Table 4.1: Classification of combining matchers

4.2 Matching Algorithms

This section describes examples of individual matchers divided by the proposed criteria.

4.2.1 Element Name Matchers

Affix matcher compares common stems (suffixes and prefixes) of element names. It takes two element names as input and checks whether the first element name starts or ends with the second one.

Example 4.1. The word `name` is a suffix of the word `surname`.

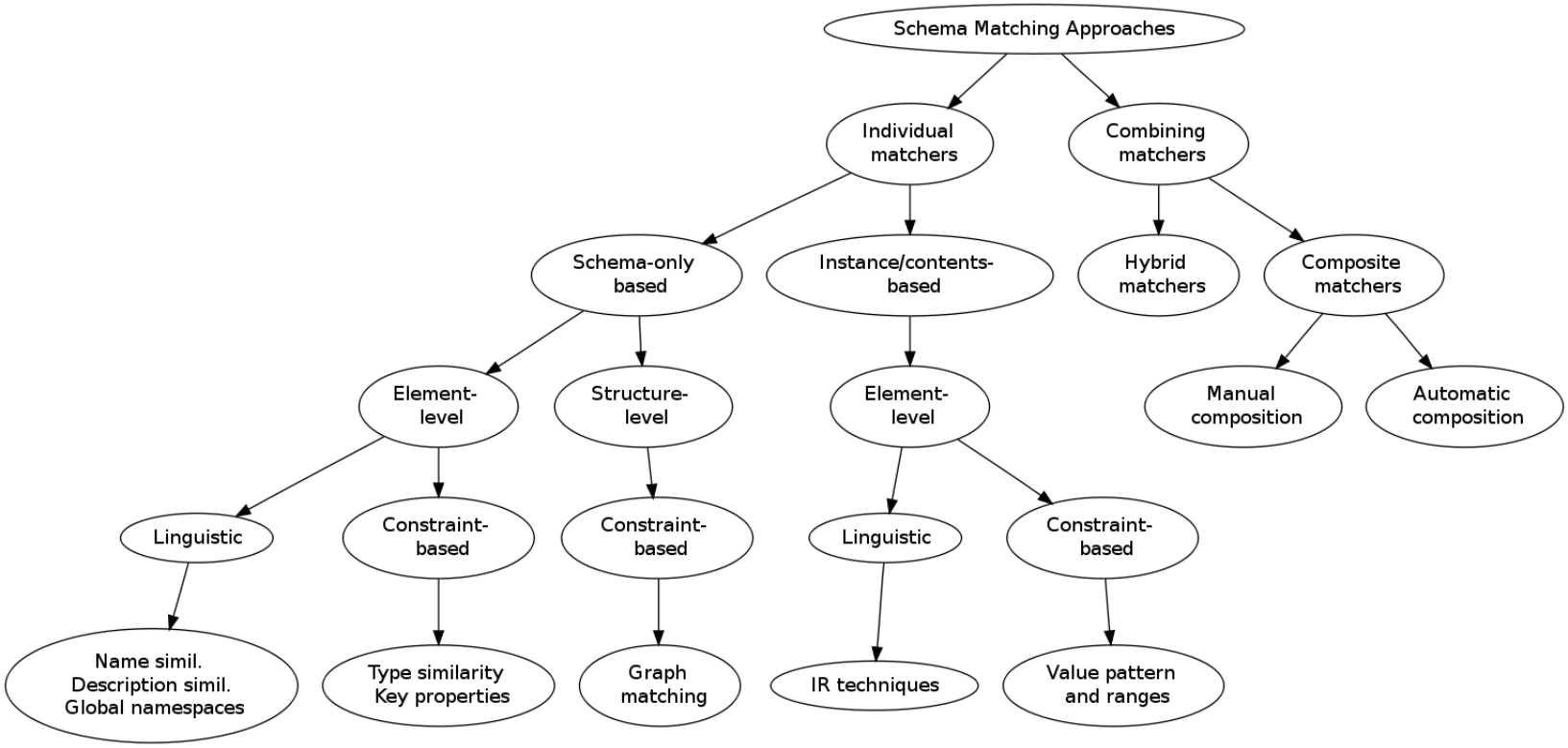


Figure 4.1: Classification of schema matching approaches proposed in [3]

N-gram matcher syntactically compares schema element names. It converts an element name into a sequence of n-grams ($n \geq 0$). An n-gram is a sequence of n characters. The number of common n-grams represents the degree of similarity between input element names. Most commonly used n-grams are with $n = 1, 2, 3, 4$.

Example 4.2. Bigrams ($n = 2$) for words `table` and `tabular` are:
table: `ta`, `ab`, `bl`, `le`
tabular: `ta`, `ab`, `bu`, `ul`, `la`, `ar`
the common bigrams are: `ta`, `ab`

The similarity score is calculated using the following formula:

$$sim(s_1, s_2) = \frac{commonNgramCount(s_1, s_2, n)}{length(s_1) + length(s_2)} \tag{4.1}$$

Distance matcher evaluates the number of edit operations necessary to transform one element name to another one. *Levenshtein* algorithm, also known as the edit distance algorithm, is a measure of the amount of difference between two strings. It is the least number of edit operations required to transform one string into another normalized by the length of the longest string. Edit operations are character *substitution*, *insertion* and *deletion*. It is possible to set the weight for each edit operation.

Example 4.3. The edit operation between word `kitten` and `kettle` are:

- k**i**tten → k**e**tten (substitution of **i** by **e**),
- kette**n** → kette (deletion of **n**),
- kette → kett**l**e (insertion of **l**).

Two equal strings do not require any transformation and have similarity score of 1. If we denote o edit operation, w_o weight of this edit operation, then the similarity score is calculated as follows:

$$editDist(s_1, s_2) = \sum_o w_o \sum editOperationCount_o(s_1, s_2) \tag{4.2}$$

$$sim(s_1, s_2) = \frac{editDist(s_1, s_2)}{max(length(s_1), length(s_2))} \tag{4.3}$$

Soundex matcher computes the phonetic similarity between names from their corresponding soundex codes. Soundex [44] indexes names by sound, as pronounced in English. An example of soundex codes is shown in Table 4.2.

Synonym matcher is a semantic matcher that consults an auxiliary source of information (dictionary) for terminological relationships between input element names. The similarity value is computed according to semantic relationship between them - whether they are *synonyms*, *hypernyms*.

Number	Represents the Letters
1	B F P V
2	C G J K Q S X Z
3	D T
4	L
5	M N
6	R

Table 4.2: Example of a Soundex codes

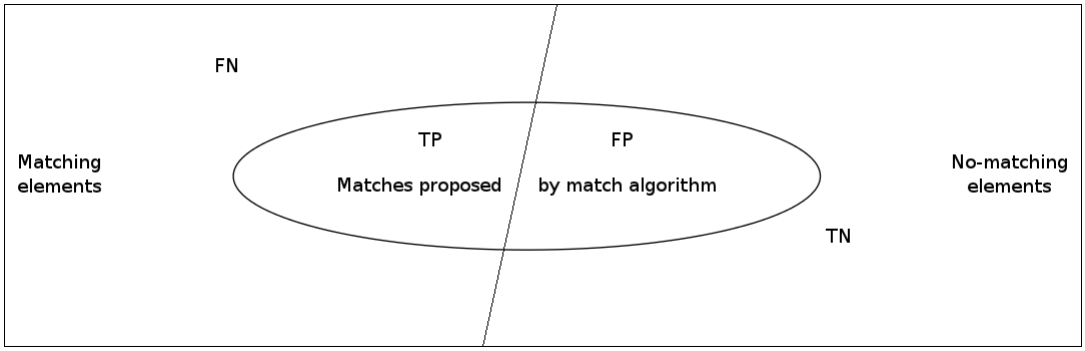


Figure 4.2: Matching algorithm decision sets

4.2.2 Structure Matchers

Tree Edit Distance algorithm is an extension of the Levenshtein algorithm for finding similarity between two trees. The basic edit operations insertion, deletion, and substitution are expanded with operations *insert tree* and *delete tree*. These operations are applied to a node of a tree. This is studied e.g. in [40].

4.3 Match Quality Measures

Several measures, such as **Recall**, **Precision**, **F-Measure** and **Overall** are used for automatic evaluation of match quality [6]. Recall and precision are metrics well known from the field of information retrieval.

During matching the algorithm has to make some decision about each pair of elements – whether elements match or do not match.

Notation 4.1. Matching decision sets:
FN - false negatives - the algorithm decided to not match elements e_i, f_j, but they do match.
TP - true positives - the decision to match elements e_i, f_j is correct.
FP - false positives - the decision of the algorithm is to match elements e_i, f_j, but they do not match.
TN - true negatives - the decision to not match element e_i to element f_j is correct.

Each of the element matching decisions falls into one of these four sets, *FN*, *TP*, *FP*, and *TN*. This is shown in Figure 4.2. Both false negatives and false positives reduce the match quality.

Recall (also called **Completeness**) indicates the proportion of matches automatically proposed by the algorithm among the complete set of elements that match. High value of recall means that the algorithm returned most of the relevant results.

$$Recall = \frac{|TP|}{|FN| + |TP|} \tag{4.4}$$

Precision (also called **Soundness**) reflects the share of correctly automatically determined matches among the set of all automatically determined matches. High value of precision means that an algorithm has high accuracy - it returned more relevant results than irrelevant.

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{4.5}$$

Both measures can be maximized at the expense of a low value of the other, but it brings no real improvement, as shown in [23]. One solution for this problem is to combine these metrics into one.

F-measure allows attaching different relative importance to Precision and Recall. Special case is for $\alpha = 1/2$ where Precision and Recall are in balance.

$$\begin{aligned} F\text{-}Measure(\alpha) &= \frac{|TP|}{(1-\alpha)|FN| + |TP| + \alpha|FP|} \\ &= \frac{Precision\ Recall}{(1-\alpha)\ Precision + \alpha\ Recall} \quad 0 \leq \alpha \leq 1 \end{aligned} \tag{4.6}$$

Overall quantifies the post-match effort needed for adding false negatives and removing false positives. This metric was proposed in [5].

$$\begin{aligned} Overall &= 1 - \frac{|FN| + |FP|}{|FN| + |TP|} \\ &= \frac{|TP| - |FP|}{|FN| + |TP|} \\ &= Recall(2 - \frac{1}{Precision}) \end{aligned} \tag{4.7}$$

4.4 Sample Approaches

In this section the best known representatives of schema matching are described.

4.4.1 COMA

COMA matcher [1] is an example of a *composite* approach. Individual matchers are selected from an extensible library of match algorithms. Matchers are *simple*, *hybrid* and *reuse-oriented*. The process of matching is *interactive* and *iterative*. A match iteration has the following three phases:

1. User feedback and selection of the match strategy,
2. Execution of individual matchers,
3. Combination of the individual match results.

Interactive mode The first step in the iteration is optional. The user is able to provide *feedback* - to confirm or reject previously proposed match candidates or to add new matches, and to define a *match strategy* - selection of matchers, strategies to combine individual match results. In *automatic mode* there is only one iteration and the match strategy is specified by input parameters.

Reuse of match results Since many schemas to be matched are very similar to the previously matched schemas, match results (intermediate similarity results of individual matchers and user-confirmed results) are stored for later reuse.

Data structure Input schemas are transformed to an internal graph representation - *rooted directed acyclic graph*. All schema elements are represented by their *full paths* from the root to a corresponding node.

Aggregation of individual matcher results Similarity values from individual matchers are aggregated to a combined similarity value. Several aggregate functions are available, for example Min, Max or Average.

Selection of match candidates For each schema element its best match candidate from another schema is selected (the ones with the highest similarity value according to MaxN, MaxDelta, Threshold criteria).

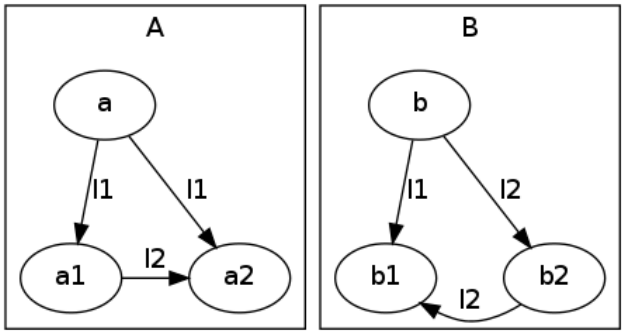
The COMA++ [15], extention of COMA, supports a number of other features like merging, saving and aggregating match results of two schemas.

4.4.2 Similarity Flooding

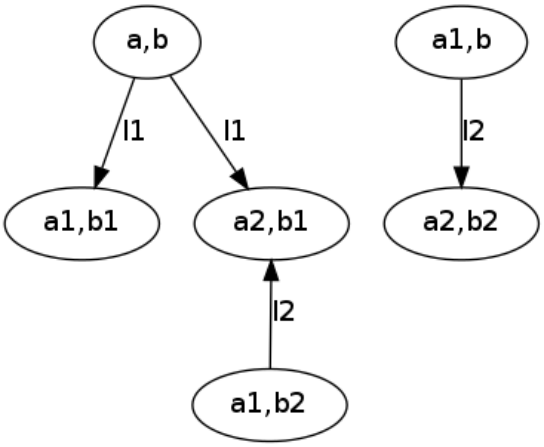
Similarity Flooding [5] can be used to match various data structures - data schemas, data instances, or a combination of both. The algorithm is based on the idea that the similarity of an element is propagated to its neighbours. The input data is converted into *directed labeled graphs*. Every edge in the graphs is represented as a triple (s, l, t), where s is a source node, t is a target node and l is a label of the edge. No external directory of terminological relationships is used. Output mapping of elements is checked and if necessary, corrected by the user. The accuracy of the algorithm is calculated as the number of needed adjustments. The algorithm has the following steps:

1. Conversion of input schemas to internal graph representation.
2. Creation of auxiliary data structures - *pairwise connectivity graph* and *propagation graph*. An example can be seen in Figure 4.3.
3. Computation of initial mapping.
4. Iterative fixpoint computation.

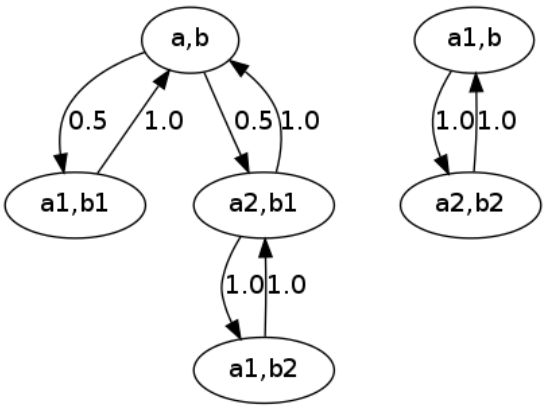
5. Selection of relevant match candidates.



(a) Input graphs



(b) Pairwise connectivity graph



(c) Propagation graph

Figure 4.3: Example of Similarity Flooding data structures

Definition 4.1. A *pairwise connectivity graph* $PCG(A, B) = (V, E)$ is a directed labeled graph, where
$A = (V_A, E_A), B = (V_B, E_B)$ are graphs representing input schemas,
$V \subseteq (V_A \times V_B)$ is set of nodes of graph $PCG(A, B)$,
$((s_A, t_A), l, (s_B, t_B)) \in E \iff (s_A, l, t_A) \in E_A \wedge (s_B, l, t_B) \in E_B$.

Definition 4.2. A *propagation graph* $PG(A, B) = (V', E')$ is a directed weighted graph induced from a pairwise connectivity graph $PCG(A, B) = (V, E)$, where
$V' = V$ is set of nodes of graph $PG(A, B)$
$((s_A, t_A), (s_B, t_B)) \in E' \wedge ((s_B, t_B), (s_A, t_A)) \in E' \iff ((s_A, t_A), l, (s_B, t_B)) \in E$
$w((s_A, t_A), (s_B, t_B)) = 0.5$
$w((s_B, t_B), (s_A, t_A)) = 1.0$
w is called propagation coefficient - the weight of the edges of propagation graph that indigates how well the similarity of map pair propagates to its neighbors.

Matcher The main matcher is structural and is used in a *hybrid* combination with a simple name matcher that compares common affixes for initial mapping. The matcher is iterative and based on *fixpoint computation* with *initial mapping* as a starting point.

Fixpoint computation The algorithm is iterated until a fixpoint has been reached (e.g. similarities stabilize).
σ^i - similarity value in i-th iteration of nodes $x \in V_A$ and $y \in V_B$
σ^0 - value computed in initial mapping

$$\begin{aligned} \sigma^{i+1}(x, y) &= \sigma^i(x, y) + \\ &+ \sum_{\substack{(a,l,x)\in E_A \\ (b,l,y)\in E_B}} \sigma^i(a, b) w((a, b), (x, y)) \\ &+ \sum_{\substack{(x,l,c)\in E_A \\ (y,l,d)\in E_B}} \sigma^i(c, d) w((x, y), (c, d)) \end{aligned} \tag{4.8}$$

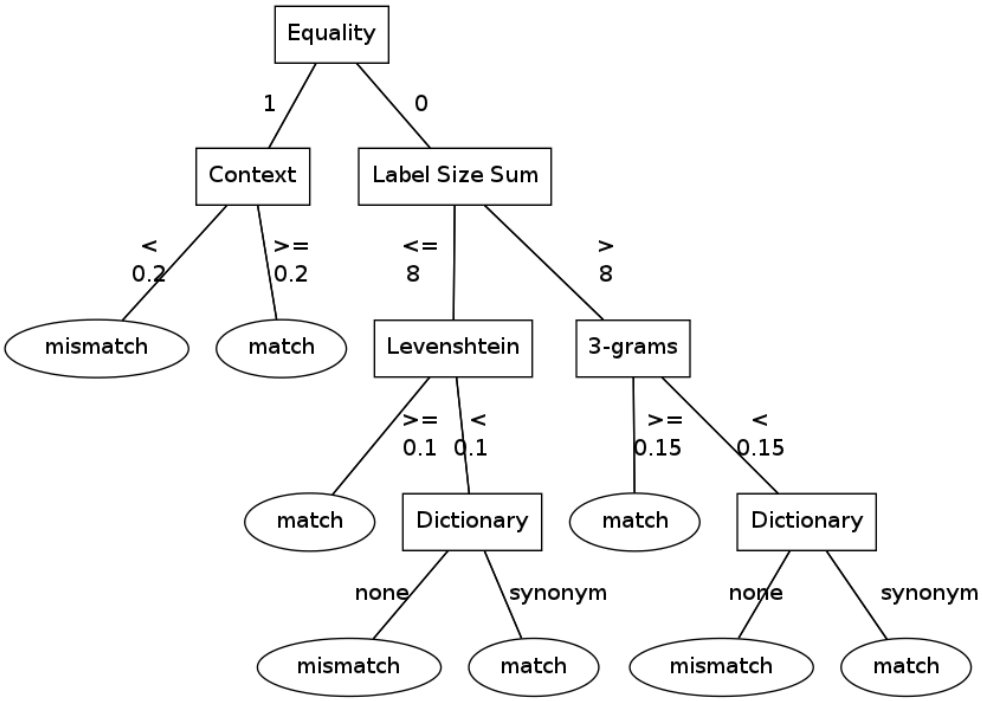
Similarity Flooding could be improved for example by usage of another matcher for initial mapping or auxiliary source of information - e.g. dictionary.

4.4.3 Decision Tree

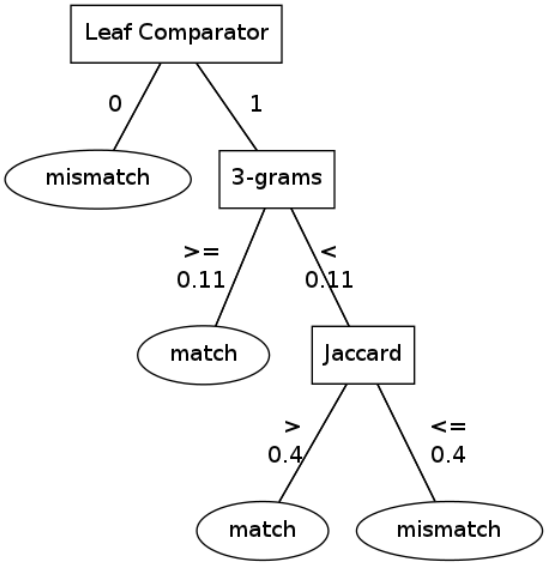
In [2] a new method of combining independent matchers was introduced. An example of a decision tree can be seen in Figure 4.4 and it is defined as follows.

Definition 4.3. A *Decision tree* is a tree $G = (V, E)$, where
V_i - set of internal nodes - independent match algorithms.
V_l - set of leaf nodes - output decision whether elements do or do not match.
$V = V_i \cup V_l$ - set of all nodes.
E - set of edges - conditions that decide to which child node the computation will continue.

Example 4.4. A decision tree that is displayed in Figure 4.4b has the following sets of nodes:
$V_i = \{$Leaf Comparator, 3-grams, Jaccard$\}$
$V_l = \{$mismatch, match, match, mismatch$\}$



(a) Quality-based



(b) Performance-based

Figure 4.4: An example of a decision tree from [2]

During the traversal of this tree we are for example at the root node where Leaf Comparator matcher can return the following similarity values:

- 0: the computation will continue to its left child that is leaf and mapping pair is identified as mismatch,
- 1: the computation will continue to its right child that is internal node and the traversal of tree will continue.

A node in a decision tree can have as many children as the similarity measure requires. For example a matcher has values from range $[a, b] >$, $a < b < c < d$ and reached value v:

- $v \in [a, b]$: *mismatch*,
- $v \in (b, c]$: *continue with computation* with another similarity measure,
- $v \in (c, d]$: *match*.

Comparison with Aggregation The decision tree approach does not have the following disadvantages of aggregation of result of independent matchers:

- **Performance** In the composite approach with an aggregate function, all of the match algorithms have to run. The time required is worse than with a decision tree.
- **Quality** Aggregation can lower the match quality, e.g. if we give higher weights to several matchers of the same type that falsely return a high similarity value.
- **Extendability** is worse, because adding a new matcher means updating the aggregation function.
- **Flexibility** is limited, because an aggregation function needs manual tuning (weights and thresholds).
- **Common threshold** Each match algorithm has its own value distribution, they should have their own threshold.

4.5 Advantages and Disadvantages

The comparison of the previously disscused methods is introduced in Table 4.3. The decision tree approach seems to be the most promising for our application – it is dynamic and versatile. Furthermore it has desirable values of compared properties – it is highly extensible, quick and has a low level of user intervention and a low level of required auxiliary information. The decision tree approach was choosen for further study and implementation in this thesis.

	Extensibility	Speed	User intervention	Auxiliary info
COMA	Low	Low	High	Low
SF	None	High	None	None
DT	High	High	Low	Low

Table 4.3: A comparison of the selected existing solutions

5. Decision Tree

This chapter contains a brief description of the algorithm for construction of decision tree proposed in the work of Jakub Stárka [22] and then follows a description of C5.0 algorithm that is used in this work for training of decision tree.

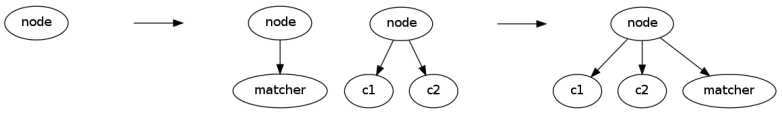
5.1 Decision Tree Induction

The decision tree in the thesis of Jakub Stárka is constructed as follows: The matchers are split into three groups according to the main feature that they compare: class name, data type and structural similarity.

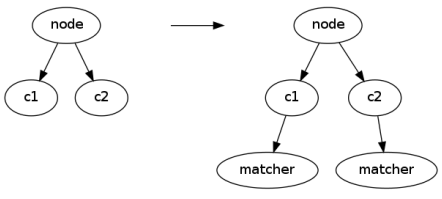
In each group the matchers are assigned with a priority according to their efficiency. Then the matchers are sorted in ascending order according to importance (where the class name group is the most important) of group and their priority inside the group. Finally the decision tree is built. The first matcher is selected as the root of the tree and other matchers are taken in sequence and added to the tree. If we want to add matcher M to the actual node n – $addMatcherToTree(M, n)$, there are the following possible situations:

- If node n has no child, method M is added as a child of n.
- If node n has children $c_1, ...c_n$ from the same feature group that M belongs to and has the same priority, then matcher M is added as the next child of node n.
- If node n has children $c_1, ...c_n$ from the different feature group that M belongs to or has different priority, then for each node c_i we call $addMatcherToTree(M, c_i)$.

These three situations are displayed in Figure 5.1.



(a) Without children (b) Same feature and priority



(c) Different feature and priority

Figure 5.1: The decision tree creation by algorithm proposed by Jakub Stárka [22]

The decision tree that is created by algorithm proposed by Jakub Stárka is displayed in Figure 5.2

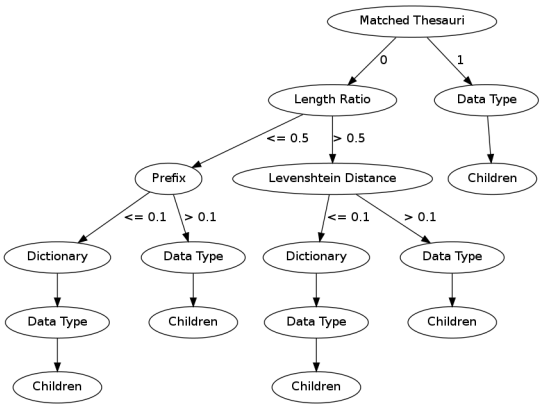


Figure 5.2: A decision tree used in work of Jakub Stárka

5.2 Decision Tree Induction via C5.0

Currently, there are several algorithms for the induction of a decision tree from training data – ID3 [16], CLS [19], CART [18], C4.5[17] and SLIQ [20]. The C5.0 [29] algorithm is exploited and utilized in this thesis. The following paragraph describes it.

In the following paragraph, we introduce a notation that is used in the rest of the chapter.

Notation 5.1. Decision tree construction:

S	– a set of training samples. An example of training samples is displayed in Figure 5.3.
$S(v, M)$	– a set of examples from S that have value v for matcher M.
$S((i_1, i_2), M)$	a set of examples from S that have value from interval (i_1, i_2) for matcher M.
$C = \{C_1, C_2\}$	– the decision tree algorithm classifies S into two subsets with possible outcomes – $C_1 = match$ and $C_2 = mismatch$.
$Info(S)$	– entropy of the set S.
$freq(C_i, S)$	– the number of examples in S that belong to class C_i.
$\|S\|$	– the number of samples in the set S.
$Gain(M, S)$	– the value of information gain for matcher M and set of samples S.
$Info_M(S)$	– entropy for matcher M.

The entropy of the set of training samples S is computed as follows (using the above defined notation):

$$Info(S) \quad = \quad -\sum_{i=1}^{2} \left(\frac{freq(C_i, S)}{|S|} \log_2 \left(\frac{freq(C_i, S)}{|S|} \right) \right) \tag{5.1}$$

The set S has to be partitioned in accordance with the outcome of matcher M. There are two possibilites:

1. Matcher M has $1..n$ discrete values, in that case the entropy for matcher M and set S is computed as follows (using the above defined notation):

$$Info_M(S) \quad = \quad \sum_{i=1}^{n} \left(\frac{|S(i, M)|}{|S|} Info(S(i, M)) \right) \tag{5.2}$$

2. Matcher M has values from continuous interval $[a, b]$, that is why threshold $t \in [a; b]$ that brings the most information gain has to be selected by Algorithm 5.2. Entropy for matcher M and set S is then computed according to the following formula (using the above defined notation):

$$Info_M(S) = \left(\frac{|S([a;t], M)|}{|S|} Info([a;t], M)\right) + \left(\frac{|S((t;b], M)|}{|S|} Info((t;b], M)\right) \tag{5.3}$$

The gain value for a set of samples S and matcher M is computed as follows:

$$Gain(M, S) \quad = \quad Info(S) - Info_M(S) \tag{5.4}$$

Then decision tree is constructed by Algorithm 5.1. There are the following possibilities for the content of the set of training samples S in the given node *parent* of the decision tree:

1. S is empty, then the decision tree is a leaf identifying class C_i – the most frequent class at the parent of the given node *parent*. This leaf is added as a child to node *parent*.

2. S contains only examples from one class C_i, then the decision tree is a leaf identifying class C_i. This leaf is added as a child to node *parent*.

3. S constains examples from different classes, then S has to be divided into subsets. The matcher M with the highest value of information gain is selected. There are two possibilities:

   (a) The matcher M has n dicrete mutually exclusive values $v_1..v_n$, then set S is partioned into subsets S_i where S_i contains samples with value v_i for matcher M.

   (b) The matcher M has values $(v_1, ...v_n)$ from continuous interval $[a, b]$, then threshold $t \in [a, b]$ has to be determined. Subsets S_1, S_2 constains samples with values from interval $[a, t]$, $(t, b]$, respectivelly, for matcher M.

   The matcher M is added as a child to node *parent*. For all the subsets S_i subtrees are constructed and are added to node M as children.

The threshold for matcher M with values $(v_1, ...v_n)$ from continuous interval $[a, b]$ is selected as follows:

- Values are sorted in the ascending order, duplicate values are removed. Lets denote them $u_1, ...u_m$.

- All possible thresholds $A_i \in [u_i, u_{i+1}]$ have to be explored.

- For each interval $[u_i, u_{i+1}]$ the midpoint A_i is choosen as a split to two subsets $[u_1, A_i]$ and $(A_i, u_m]$.

- For each midpoint the information gain is computed and the midpoint A_{max} with the highest value of information gain is selected .
- The threshold is then returned as a lower bound of interval $[u_{max}, u_{max+1}]$.

Algorithm 5.1 Construction of a decision tree T from a set S of user-evaluated training samples

```
1: function BuildTree(S, parent, condition)
2:     T empty tree
3:     if S is empty then
4:         c ← the most frequent class at the parent of the given node parent
5:         /* adds node c as a child to node parent with condition condition
6:         on edge */
7:         AddLeaf(parent, c, condition)
8:     else if S constains only results from one class C_i then
9:         c ← C_i
10:        /* adds node c as a child to node parent with condition condition
11:        on edge */
12:        AddLeaf(parent, c, condition)
13:    else
14:        M ← matcher with the highest value of information gain Gain(M, S)
15:        /* adds node M as a child to node parent with condition condition
16:        on edge */
17:        AddNode(parent, M, condition)
18:        if M has n dicrete mutually exclusive values v_1..v_n then
19:            S' ← {S_1, ...S_n}|S_i = S(v_i, M)
20:            c_i ← v_i
21:        else if M has values (v_1, ...v_n) from continuous interval [a, b] then
22:            t ← ComputeTreshold((v_1, ...v_n), M, S)
23:            /* samples with values from interval [a, t] for matcher M */
24:            S_1 ← S([a, t], M)
25:            c_1 ← [a, t]
26:            /* samples with values from interval (t, b] for matcher M */
27:            S_2 ← S((t, b], M)
28:            c_2 ← (t, b]
29:            S' ← {S_1, S_2}
30:        end if
31:        for all S_i ∈ S' do
32:            /* constructs subtree T_i from subset S_i and adds it as a child to
33:            node M with condition c_i on edge */
34:            T_i ← BuildTree(S_i, M, c_i)
35:        end for
36:    end if
37:    return T
38: end function
```

Algorithm 5.2 Selection of threshold for continuous values $v_1, ...v_n$ for matcher M and set of samples S

```
1: function ComputeTreshold((v_1, ...v_n), S, M)
2:     /* sorts values in the ascending order, duplicate values are removed */
3:     (u_1, ...u_m) ← SortAscDistinct((v_1, ..v_n))
4:     for i ← 1, m − 1 do
5:         /* average of values U[i] and U[i+1] */
6:         A[i] ← Avg(U[i], U[i + 1])
7:         L[i] ← U[i]
8:         H[i] ← U[i + 1]
9:     end for
10:    /* u_m the highest value from the continuous inteval */
11:    /* u_1 the lowest value from the continuous interval */
12:    for i ← 1, m − 1 do
13:        gain_i ← Gain(M, S([u_1, A[I]], (A[I], u_m], M))
14:    end for
15:    maxGain ← max_{i=1}^{m−1} gain_i
16:    max ← i|gain_i = maxGain
17:    t ← L[max]
18:    threshCost ← cost of splitting interval into two subintervals
19:                 [u_1, t] and (t, u_m]
20:    result.gain ← maxGain − threshCost
21:    result.threshold ← t
22:    return result
23: end function
```

Example 5.1. An example of training of the decision tree using the C5.0 algorithm: For briefness only three matchers are used: `Matched Thesauri`[1], `Levenshtein Distance` and `N-gram`. In Figure 5.1 a small training set is shown. C5.0 algorithm works in the following steps:

- In the begining, the training set S contains all 14 samples. `Matched Thesauri` has discrete values 0 and 1. `Levenshtein Distance` and `N-gram` have values from continuous interval $[0; 1]$. Gain values are computed for all matchers.

  `Matched Thesauri` has the highest gain value – 0.371, that is why `Matched Thesauri` is selected as a root of the constructed decision tree. Set S is divided into two parts $S(0, \texttt{Matched Thesauri})$ and $S(1, \texttt{Matched Thesauri})$.

- Set $S(1, \texttt{Matched Thesauri})$ contains samples that have value 1 for matcher `Matched Thesauri` and it contains only samples from the same match class. New leaf **match** is added as a child to node `Matched Thesauri`.

- Set $S_{MT0} = S(0, \texttt{Matched Thesauri})$ consists of samples with value 0 for matcher `Matched Thesauri` and results from various classes, so this set has to be further divided. Gain values are computed and matcher with the highest gain value – `N-gram` is added as a child of node `Matched Thesauri`.

[1]Matched Thesauri matcher is described in Section 6.2

Threshold value for `N-gram` matcher with continous range is 0.071 and set S_{MT0} is devided into two subsets $S_{N1} = S([0; 0.071]$, `N-gram`) and $S_{N2} = S((0.071; 1]$, N-gram). This split is displayed in Figure 5.4.

- There are only mismatch results in set S_{N1}, so leaf **mismatch** is added as a child to node `N-gram`.
- Set S_{N2} also contains results from one class – match. Another leaf **match** is added to node `N-gram`.

The final trained decision tree is displayed in Figure 5.5.

The algorithm proposed by Jakub Stárka has several possible drawbacks. It does not propose a method for automatic determination of conditions on edges and thresholds for continuous matchers. They have to be either set by the user or the default values are used. Furthermore, the decision tree does not suggest mapping results automatically. It computes an aggregated similarity score. During the traversal of the decision tree for each of the feature groups the maximal similarity value returned by matcher from this group is stored. Then the aggregated similarity score is computed as an average of the maximal similarity value for each of the feature groups. For each PSM element it returns possible match candidates – PIM elements evaluated by the aggregate similarity score sorted in the descending order. This helps to find matches, but it is not done automatically – the user has to evaluate each mapping. Because of this, we are not able to compare the results of our approach with the previous one.

In this work we decided to generate the decision tree using machine learning techniques. This approach solves the above mentioned problems, as we would like to use the advantages of the decision tree approach and minimize the previous disadvantages.

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
1	0	0	match
0	0.167	0	mism
0	0.1	0	mism
1	0.2	0.063	match
1	0	0	match
0	0	0	mism
0	0.1	0	mism
0	0.5	0.25	match
0	0	0	mism
0	0.1	0	mism
0	0.429	0.25	match
0	0.125	0.071	mism
0	0.6	0.385	match
1	0.2	0.067	match

	match	mism	0.183	match	mism	0.032	match	mism	
0	3	7	<=	2	7	<=	2	6	
1	4	0	>	5	0	>	5	1	
		0.371			0.324			0.115	

Matched Thesauri = 1

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
1	0	0	match
1	0.2	0.063	match
1	0	0	match
1	0.2	0.067	match

Matched Thesauri = 0

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
0	0.167	0	mism
0	0.1	0	mism
0	0	0	mism
0	0.1	0	mism
0	0.5	0.25	match
0	0	0	mism
0	0.1	0	mism
0	0.429	0.25	match
0	0.125	0.071	mism
0	0.6	0.385	match

	match	mism	0.298	match	mism	0.161	match	mism	
0	3	7	<=	0	7	<=	0	7	
1	0	0	>	3	0	>	3	0	
		0			0.762			0.781	

Figure 5.3: Data used for training of decision tree in Example 5.1

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
0	0.167	0	mism
0	0.1	0	mism
0	0	0	mism
0	0.1	0	mism
0	0.5	0.25	match
0	0	0	mism
0	0.1	0	mism
0	0.429	0.25	match
0	0.125	0.071	mism
0	0.6	0.385	match

	match	mism	0.298	match	mism	0.161	match	mism
0	3	7	<=	0	7	<=	0	7
1	0	0	>	3	0	>	3	0
		0			0.762			0.781

N-gram > 0.071

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
0	0.5	0.25	match
0	0.429	0.25	match
0	0.6	0.385	match

N-gram <= 0.071

Matched Thesauri	Levenshtein Distance	N-gram	
Discrete	Continuous	Continuous	
0	0.167	0	mism
0	0.1	0	mism
0	0	0	mism
0	0.1	0	mism
0	0	0	mism
0	0.1	0	mism
0	0.125	0.071	mism

Figure 5.4: Data used for training of decision tree in Example 5.1

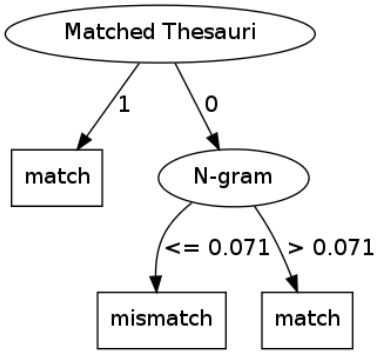


Figure 5.5: Decision tree trained from the training set in Example 5.1

6. Implementation

This chapter contains the implementation details. There is description of the tool in which this thesis was implemented. It also contains a survey of used match algorithms and describes the user interface.

6.1 Platform

This thesis is implemented in the eXolutio [41] tool. eXolutio is based on the MDA approach and models XML schemas at two levels – PIM and PSM. eXolutio allows the user to manually design a common PIM schema and multiple PSM schemas with interpretations against the PIM schema. Mapping between the two levels allows to propagate a change to all the related schemas. Thesis of Jakub Stárka has been implemented within eXolutio predecessor – XCase[1]. Complete information, documentation and download possibilities for eXolutio can be found at the project web page[2].

6.2 Set of Used Matchers

This thesis uses the same set of matchers as the one of Jakub Stárka: `Matched Thesauri`, `Length Ratio`, `Levenshtein Distance`, `Prefix`, `Data Type` and `Children`. The set of matchers could be easily extended, as described in Section 6.4.

`Matched Thesauri` Thesaurus for this matcher contains user-confirmed matches from the previous matchings. Names of matched classes or attributes (s_1, s_2) are compared with the previous user-confirmed match pairs and evaluated by the following score:

$$matchedThesauri(s_1, s_2) = \begin{cases} 0 & \text{if } (s_1, s_2) \text{ is not in thesaurus} \\ synonymCoef & \text{if } (s_1, s_2) \text{ is in thesaurus} \end{cases} \tag{6.1}$$

The user is enabled to set the value of $synonymCoef \in (0; 1]$. The default value is set to 1. Thesaurus with user-confirmed matches can be saved for each mapping, so the post-match effort for adding false negatives and removing false positives is not wasted.

`Length Ratio` computes the ratio between lengths of two input strings s_1, s_2 (class or attribute name) and is defined as follows:

$$lengthRatio(s_1, s_2) = \frac{min(length(s_1), length(s_2))}{max(length(s_1), length(s_2))} \tag{6.2}$$

[1] `www.ksi.mff.cuni.cz/xcase`
[2] `http://exolutio.com/`

`Levenshtein Distance` has been described in Section 4.2. The user is enabled to set the following coefficients:

- w_s – weight of the operation substitution,
- w_d – weight of the operation deletion,
- w_i – weight of the operation insertion.

`Prefix` compares whether the string s_1 is a prefix for the string s_2 or the other way around. Names of classes and attributes (s_1, s_2) are tokenized into a sequence of tokens $s_1 = (a_1, ...a_n)$ and $s_2 = (b_1, ...b_m)$. Then the similarity value is computed as follows:

$$prefixCount(s_1, s_2) = |\{(a_i, b_j)|a_i = startsWith(b_j) \vee b_j = startsWith(a_i)\}| \tag{6.3}$$

$$prefix(s_1, s_2) = \frac{prefixCount(s_1, s_2)}{max(n, m)} \tag{6.4}$$

`N-gram` has been described in Section 4.2. The user is enabled to set the coefficient N – the length of N-grams.

`Dictionary` The domain thesaurus for this matcher contains a set of synonyms or abbreviations common for the given domain. It helps to identify matches that would be otherwise difficult to obtain. The similarity value for `Dictionary` matcher is computed as follows:

$$dictionary(s_1, s_2) = \begin{cases} 0 & \text{if } (s_1, s_2) \text{ is not in thesaurus} \\ synonymCoef & \text{if } (s_1, s_2) \text{ is in thesaurus} \end{cases} \tag{6.5}$$

`Data Type` compares similarity of data types of the given elements. It consists of the data type name and the data type relation and it is computed from data type hierarchy tree which is displayed in Figure 6.1.

Notation 6.1. Notation in data type formula:

- d – a depth of the node that is the common parent of both nodes dt_1 and dt_2 in the data type tree.
- l - a length of the shortest path between nodes dt_1 and dt_2 in data type tree.
- α, β – correction coefficients, by default they are set to values $\alpha = \beta = 0.3057$, as suggested in [47].

The similarity value for `Data Type` matcher is computed as follows:

$$dataType(dt_1, dt_2) = \begin{cases} 1 & dt_1 = dt_2 \\ e^{-\beta l} \times \frac{e^{\alpha d} - e^{-\alpha d}}{e^{\alpha d} + e^{-\alpha d}} & dt_1 \neq dt_2 \end{cases} \tag{6.6}$$

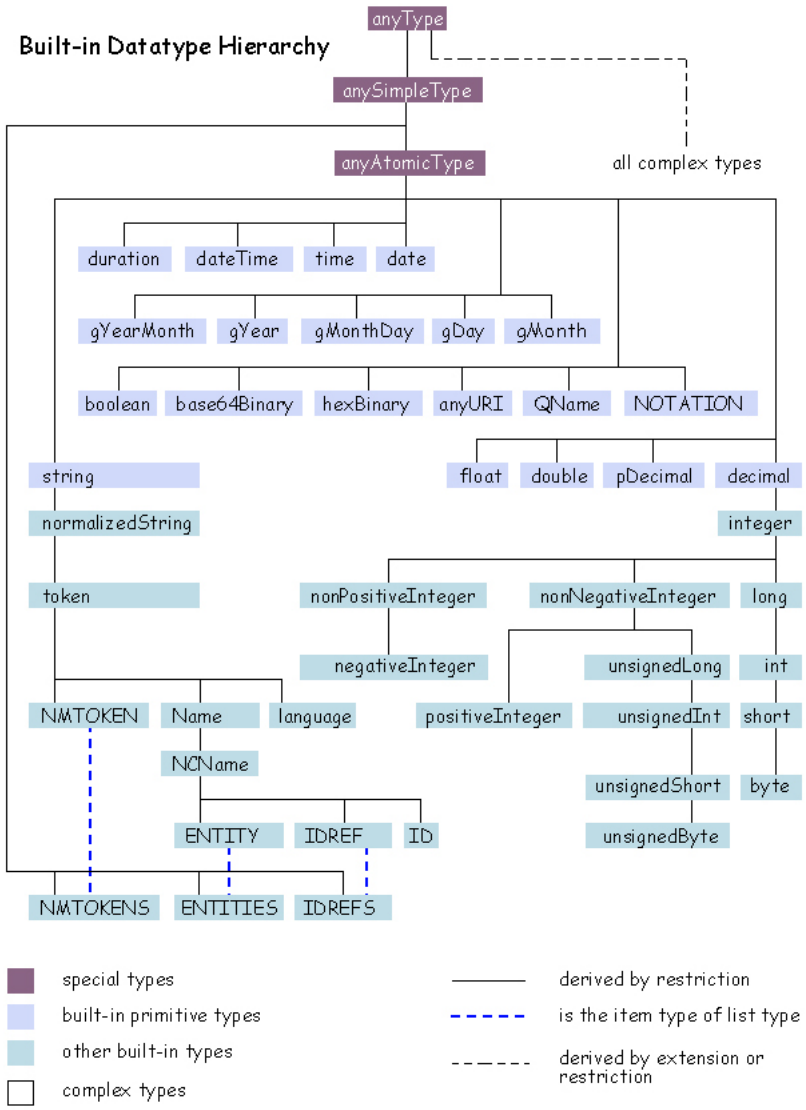


Figure 6.1: Hierarchy of XML Schema types [45]

`Children` compares the structural similarity of child nodes or neighbouring nodes of classes. The distance between classes $class_1$ and $class_2$, $dist(class_1, class_2)$, is defined as the shortest path between them. Then $d(class)$ is the total distance between class and all its children (or neighbours):

$$d(class_{psm}) = \sum dist(class_{psm}, child_{psm}) \tag{6.7}$$

$$d(class_{pim}) = \sum dist(class_{pim}, representative(child_{psm})) \tag{6.8}$$

The `Children` matcher similarity value between $class_{psm}$ and $class_{pim}$ is computed as the ratio of distances to their children:

$$children(class_{psm}, class_{pim}) = \frac{min(d(class_{psm}), d(class_{pim}))}{max(d(class_{psm}), d(class_{pim}))} \tag{6.9}$$

In Table 6.1 ranges for continuous and possible values for used matchers are shown.

Matcher	Range	Values
Matched Thesauri	Discrete	0;1
Length Ratio	Continuous	[0;1]
Data Type	Continuous	[0;1]
Dictionary	Discrete	0;1
Children	Continuous	[0;1]
Levenshtein	Continuous	[0;1]
N-gram	Continuous	[0;1]
Prefix	Continuous	[0;1]

Table 6.1: Ranges of used matchers

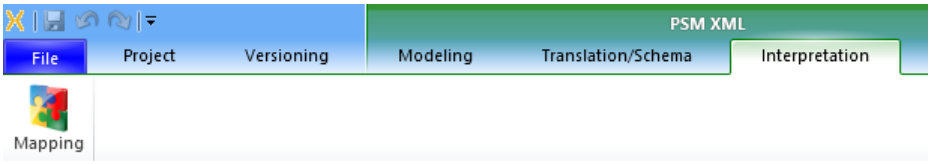


Figure 6.2: eXolutio menu for the Adaptive Similarity of XML Data module

6.3 User Interface

A detailed description of the user interface can be found in document User Documentation on DVD enclosed to this thesis. This section provides just a short description.

The module for Adaptive Similarity of XML Data is launched from **eXolutio**. The user has to open an eXolutio project and select a PSM schema. In the menu (see Figure 6.2) there is ribbon **Interpretation** → **Mapping** that displays module for matching. It has the following tabs:

Setting The user is enabled to load domain thesauri for matcher `Dictionary` and results of previous matches for matcher `Matched Thesauri`. Afterwards it is possible to set the measures used for evaluation of matching (Precision, Recall, F-Measure, Overall), select matchers that will be used for matching, view and edit their properties and coefficients.

Prepare for Training In this tab, XML schemas are prepared for training – they are converted to PSM schemas and for each pair of elements similarity values of selected matchers are computed. The user has to annotate match pairs – whether they match or mismatch. This could be tedious work and the interface is designed for maximal possible reuse of already annotated values in different scenarios, because it is possible, for example, to train a separate tree for classes and attributes or change a set of selected matchers.

Train DT The previously prepared training set is transformed to the format required by C5.0 algorithm and then tree is trained. The decision tree is displayed using the GraphViz [14] tool.

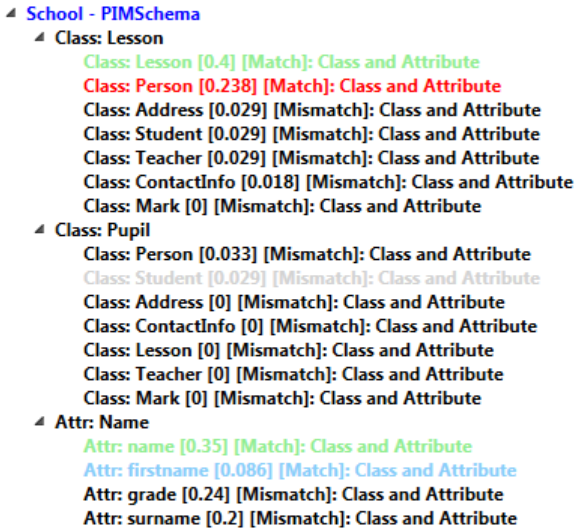


Figure 6.3: Mapping result

Mapping Mapping is the main tab that maps selected XML schemas to PIM schema. The results of matching are displayed in a tree view as shown in Figure 6.3. After matching they have the following colors:

- Blue – matches suggested by matcher, but not yet confirmed by the user,
- Black – mismatches suggested by the algorithm.

Afterwards the results can be evaluated by the user – by clicking on a match pair he confirms or declines suggested mapping result. The results are accordingly recolored:

- Green – true positives,
- Red – false positives,
- Grey – false negatives,
- Black – true negatives.

Mapping Results Tab Mapping Results displays the results of the previous matching in tree view and enables the user to confirm or deny matches suggested by the algorithm, check, correct or re-evaluate them. It also displays match quality measures values.

6.4 Extending the Set of Matchers and Set of Mapping Quality Measures

The module for Adaptive Similarity of XML Data was designed with a possible extention in mind. Sets of matchers and measures are easily extendable.

New Matcher A new matcher has to be inherited from the parent class `Matcher`:

```
public abstract class Matcher
{
  public Matcher(int id, String name, MatcherRange type)
  {
     // initialize all values
  }
  // id of matcher
  public int Id { get; set; }
  // name of matcher
  public String Name { get; set; }
  // flag whether matcher is selected for mapping
  public Boolean IsSelected { get; set; }
  // continuous or discrete range of matcher
  public MatcherRange Range { get; set; }
  // coefficients of this matcher
  public Coefficients Coefficients { get; set; }
  // all possible values for discrete matcher
  public DiscreteValues DiscreteValues { get; set; }

  // computation of similarity values
  // for all possible element types
  public abstract double ComputeSimilarity(String input1,
                                           String input2);
  public abstract double ComputeSimilarity(PSMClass input1,
                                           PIMClass input2);
  public abstract double ComputeSimilarity(PSMClass input1,
                                           PSMClass input2);
  public abstract double ComputeSimilarity(PSMAttribute input1,
                                           PIMAttribute input2);
  public abstract double ComputeSimilarity(PSMAttribute input1,
                                           PSMAttribute input2);
}
```

We need to implement all the abstract methods for a new matcher class `NewMatcher`.

```
public class NewMatcher : Matcher
{
  public NewMatcher(): base(newMatcherID,
                    "Name of new matcher",
                     newMatcherRange)
  {

    // addition of possible coefficients
    Coefficients.AddCoefficient(initValue, minValue, maxValue);
  }
```

```
  // implementation of all abstract methods
}
```

We add a new matcher to set of all the matchers:

```
commonSetting.AddNewMatcher(new NewMatcher())
```

New Mapping Quality Measure Computation of quality measure is represented by the following interface:

```
interface IComputableMeasure {
  // Computes value of match quality measure
  double ComputeMeasure(
                        // result of mapping to be evaluated
                        // by measure
                        MappingResult result,
                        // coefficient for measure
                        double coefficient,
                        // Type of used decision tree
                        // (common, separate for classes,
                        // separate for attributes)
                        DTType type);
}
```

A common parent for all quality measures is:

```
public class MatchQualityMeasure
{
  public MatchQualityMeasure()
  {
    Name = "";
  }

  // Name of the measure
  public String Name { get; set; }
  // Computation of measure
  public IComputableMeasure Computable { get; set; }
  // Computes measure
  public double Compute(
                        // Result of mapping to be evaluated
                        // by measure
                        MappingResult result,
                        // Coefficient for measure
                        double coefficient,
                        // Type of used decision tree
                        // (common, separate for classes,
                        // separate for attributes)
                        DTType type)
  {
```

```
    return Computable.ComputeMeasure(result, coefficient, type);
  }
}
```

For a definition of a new match quality measure we need to implement class for its computation:

```
class ComputeNewMeasure : IComputableMeasure {
  double ComputeMeasure(MappingResult result,
                        double coefficient,
                        DTType type)
  {
      double result = 0.0;
      // implementation of computation of new measure
      return result;
  }
}
```

Then we define `NewMeasure` class as follows:

```
class NewMeasure : MatchQualityMeasure
{
  public NewMeasure()
  {
    Name = "New Measure Name";
    Computable = new ComputeNewMeasure();
  }
}
```

We have to add a new measure to set of all the measures:

```
commonSetting.AddNewMeasure(new NewMeasure())
```

7. Experiments

This chapter contains description of all experiments and their results. The work of Jakub Stárka does not contain a wide range of experiments and one of our goals and contribution is experimental evaluation of the proposed approach.

7.1 Experimental Setup

All experiments were run on a standard personal computer with the following configuration:

```
Intel(R) Core(TM) i5-3470 3.20 GHz processor
8 GB RAM
OS 64-bit Windows 7 Home Premium SP1
```

Data for Training The following sets of XML schemas have been used for training of the decision tree:

- `BMEcat` is a standard for exchange of electronic product catalogues[1].
- `OpenTransAll` is a standard for bussiness documents[2].
- `OTA` focuses on the creation of electronic message structures for communication between the various systems in the global travel industry[3].

PIM schemas A PIM schema was designed for this thesis according to the motivation in Section 1.1 and describes a common interface for planning various types of holidays. It can be seen in Figure B.4.

XML Schemas for experiments For evaluation the following XML schemas were used:

- Artificial XML schemas: `01_Hotel`[4]
- Realistic XML schemas `02_HotelReservation`[5], `03_HotelAvailabilityRQ`[6]

These schemas are displayed in Figures B.1, B.2 and B.3.

Domain Thesaurus The domain thesaurus contains sets of words that are related semantically – for example they are synonyms or abbreviations common for the given domain. The user is enabled to expand the following one or create a completly new thesaurus. The thesauri are used during matching by `Dictionary` matcher. In Table 7.1 domain thesaurus for the domain of hotels is shown. Each row in the table contains words that are related semantically.

[1] `www.bmecat.org`
[2] `www.opentrans.de`
[3] `www.opentravel.org`
[4] This schema was designed for purpose of this work.
[5] `http://kusakd5am.mff.cuni.cz/hb/schema/reservation`
[6] `http://itins4.madisoncollege.edu/IT/152121advweb/XMLExamples/unit3/schemaSimple/HotelAvailabilityRQ.xsd`

address location
accomodation hotel
boarding meal
count amount
lengthOfStay numberOfNights

Table 7.1: Domain thesaurus for Holiday Planning: Hotel

7.2 Description of Experiments and Results

This section contains the description of experiments and presents their results. The results of the mapping are evaluated by match quality measures (see in Section 4.3) and comparison of their values is displayed in graphs generated by GnuPlot [13]. For each experiment, a short example of mapping results for all the element pairs is attached.

7.2.1 Separate Decision Trees and Common Decision Tree for Classes and Attributes

The first experiment is designed from the following observation: Efficiency of methods used to measure similarity between elements depends on the type of elements - if they are **classes** or if they are **attributes**. In this experiment two sets of decision tree are used:

1. Two separate trees for classes and for attributes.
2. One common tree for classes and attributes.

```
Experiment Setup:
  Separate Decision Trees and Common Decision Tree for Classes
  and Attributes (SeparateTrees)

  eXolutio Project:
   Travelling
  XSD schemas:
    01_Hotel
    02_HotelReservation
    03_HotelAvailabilityRQ
  Decision tree:
    1. Separate decicion tree
         for attributes (in Figure 7.1)
         for classes  (in 7.2)
    2. Common decision tree (in Figure 7.3)
  Decision tree training set:
    Set of XSD Schemas: OTA
    Sample count:
      Separate decision tree for attributes: 27 942 match pairs
```

```
    Separate decison tree for classes: 27 793 match pairs
    Common decision tree: 55 815 match pairs
Thesaurus for Dictionary:
  None
Thesuarus for Matched Thesauri:
  None

Matchers:
  Children
  DataType
  Dictionary
  Length Ratio
  Levenshtein Distance
  Matched Thesauri
  Prefix
```

In the first experiment no additional source of information was used. Both sets of decision trees were trained without domain thesauri and without previous user matches. Both sets of decision trees are induced from the same set of training samples `OTA`, particularly match pairs of XML schema `OTA_HotelAvailGetRQ.xsd` and XML schema `OTA_HotelAvailGetRS.xsd`. A separare decision tree for classes and attributes is trained only from match pairs of classes and attributes respectively. The common tree is trained from both sets together. The final decision trees are shown in Figures 7.1, 7.2 and 7.3.

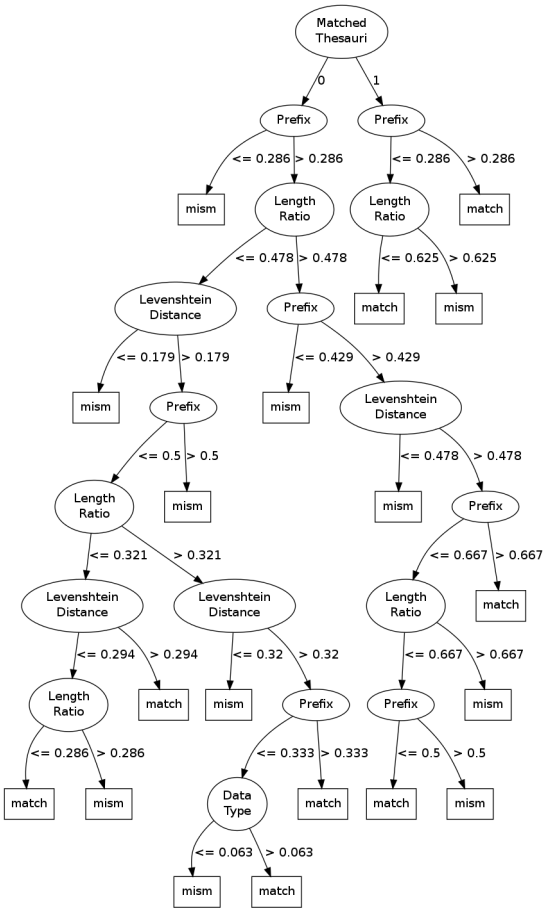


Figure 7.1: A separate decision tree for attributes for experiment *SeperateTrees*

The root of the separate decision tree for attributes is `Matched Thesauri`, all the other trees in this experiment have `Levenshtein Distance`. The matcher at the second level is the same for both branches and they have the same threshold. Interesting is the subtree for mapping pairs that are contained in `Matched Thesauri`. We would assume that this subtree should be smaller or even a leaf with the value 'match'. This could be explained by errors in user annotation of mapping results – that same match pair is annotated with different matching decision then the previous one or some mapping pairs have different meaning in different context.

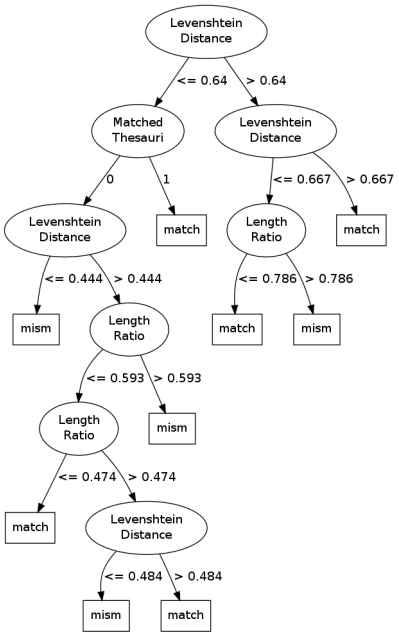


Figure 7.2: Separate decision tree for classes for experiment *SeparateTrees*

The separate decision tree for classes is relatively simple. It contains only matchers `Levenshtein Distance`, `Matched Thesauri` and `Lenght Ratio`, other matchers are not used. It corresponds with the original observation that some methods are more effective for certain types of elements. Matchers whose similarity values do not distinguish mapping pairs enough are not included. Pairs that are contained in the thesaurus are directly suggested as matches.

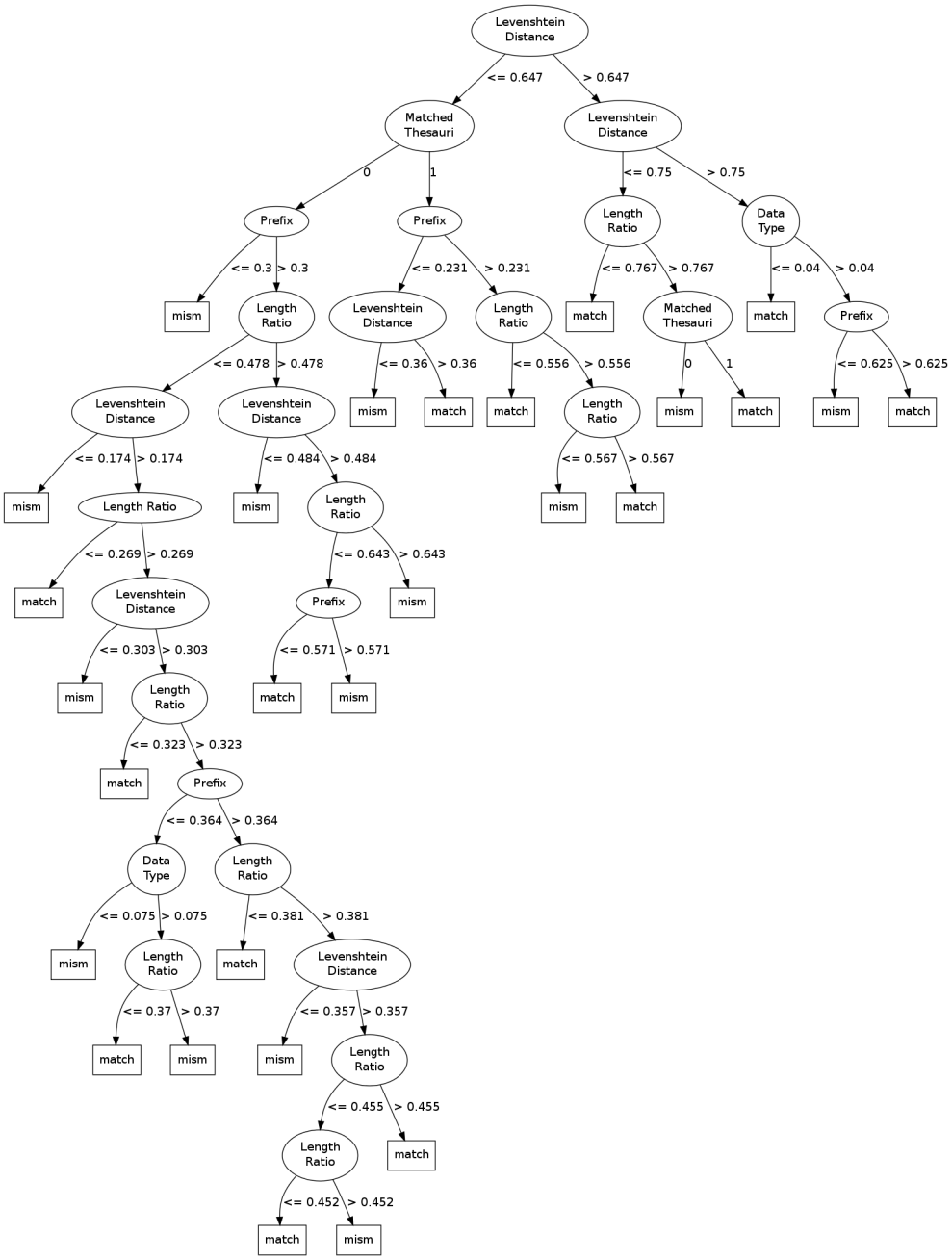


Figure 7.3: Common decision tree for experiment *SeparateTrees*

The threshold value for `Matched Thesauri` matcher in the root of the common tree and the separate tree for classes is nearly similar. The common decision tree is the most complex from the above mentioned. The common tree also contains two subtrees for `Matched Thesauri`. The first one is at the second level and it contains two full subtrees for both the values. The right subtree for pairs that are contained in thesauri is more complex than the tree in the separate tree for attributes. This could be caused by a larger number of training samples that allows for higher resolution. The second `Matched Thesauri` is directly a parent of the leaves.

In Figures 7.4, 7.5, 7.6 and 7.7 there are displayed the histograms of the match quality measures – Precision, Recall, F-Measure and Overall respectively. All the measures are at first computed for both types of elements together and

then separately for attribute and class elements.

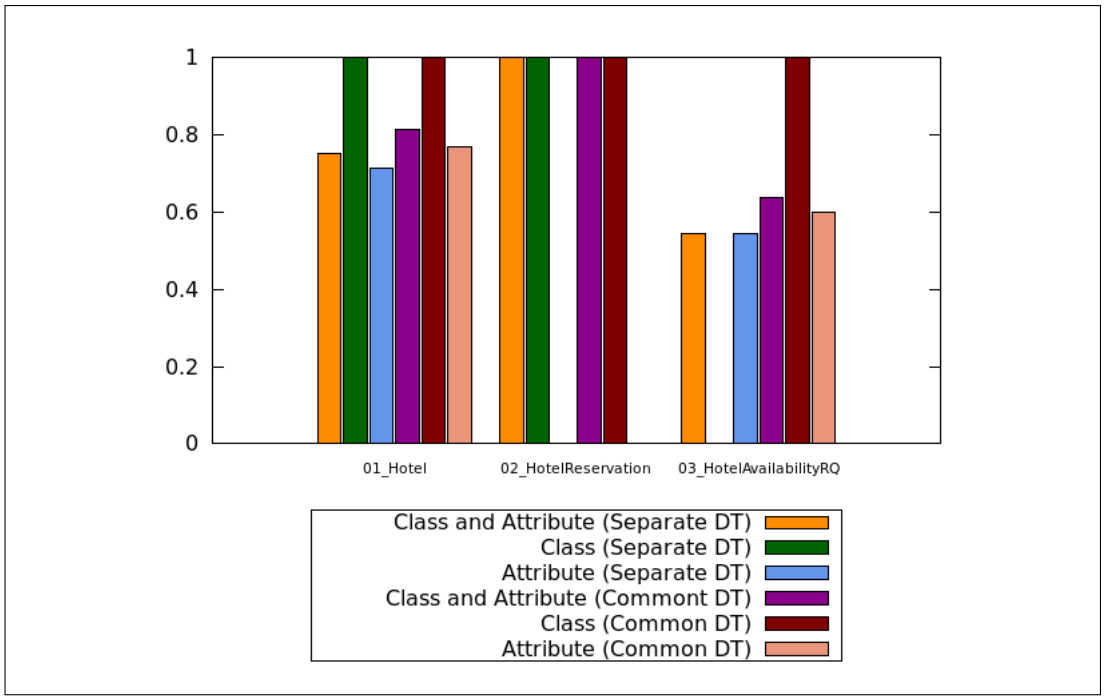


Figure 7.4: Precision for experiment *SepearateTrees*

In Figure 7.4 Precision is high for classes in all schemas and for both types of trees. The quality of mapping decision differs significantly with the type of element, but the training set contains a similar number of match pairs for classes (27 793 match pairs) and attributes (27 942 match pairs). The separate tree for classes did not suggest any mapping pair as a match for schema `03_HotelAvailabilityRQ`, just as the separate tree for attributes for schema `02_HotelReservation`. Attributes in schema `03_HotelAvailabilityRQ` are difficult to identify for all the decision trees. All Precision values are from the interval [0.545; 0.769] – they identified almost the same number of relevant results as irrelevant.

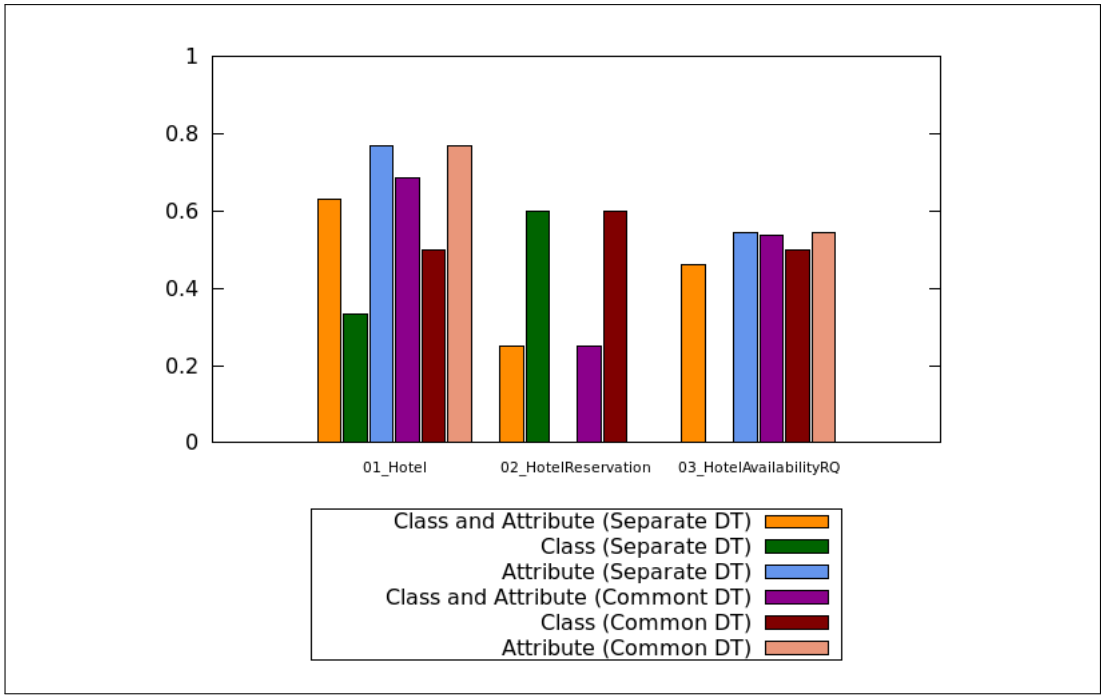


Figure 7.5: Recall for experiment *SeparateTrees*

Recall is lower then Precision in all the cases except for schema `01_Hotel` and the separate tree for attributes in Figure 7.5. There were no true positives attributes for schema `02_HotelReservation` for both trees and no true positives classes for schema `03_Hotel- AvailabilityRQ` for separate tree. Values of Recall are lower for attributes than Recall for classes.

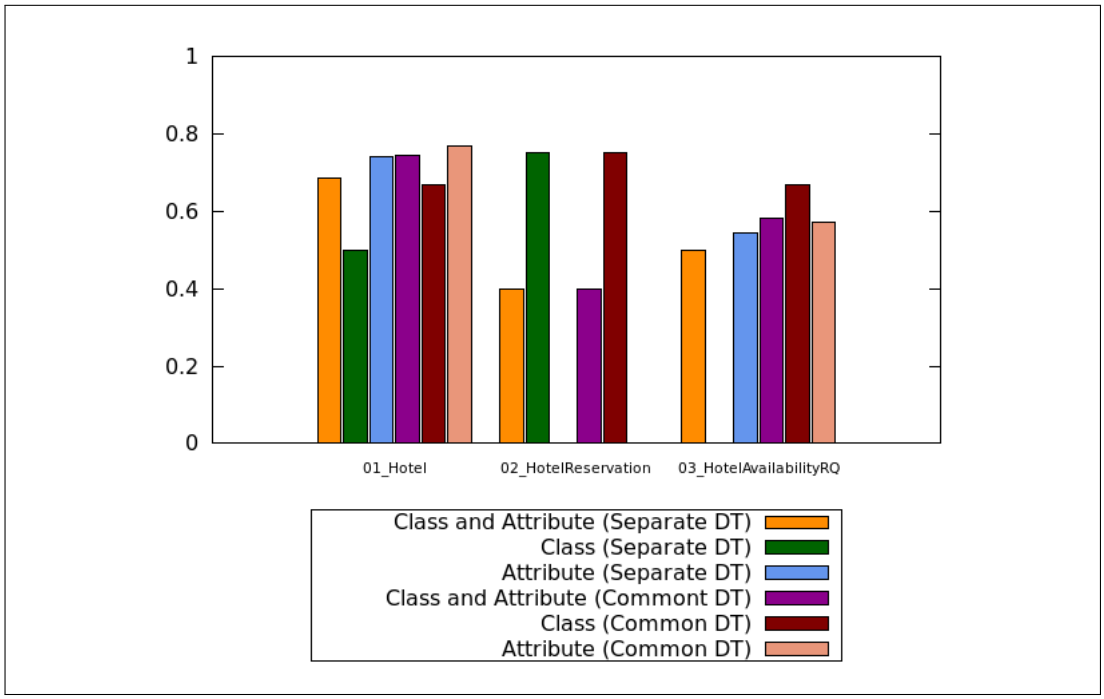


Figure 7.6: F-Measure for experiment *SeperateTrees*

In Figure 7.6 the values for F-Measure are equal for schema `02_HotelReservation` for classes for both trees.

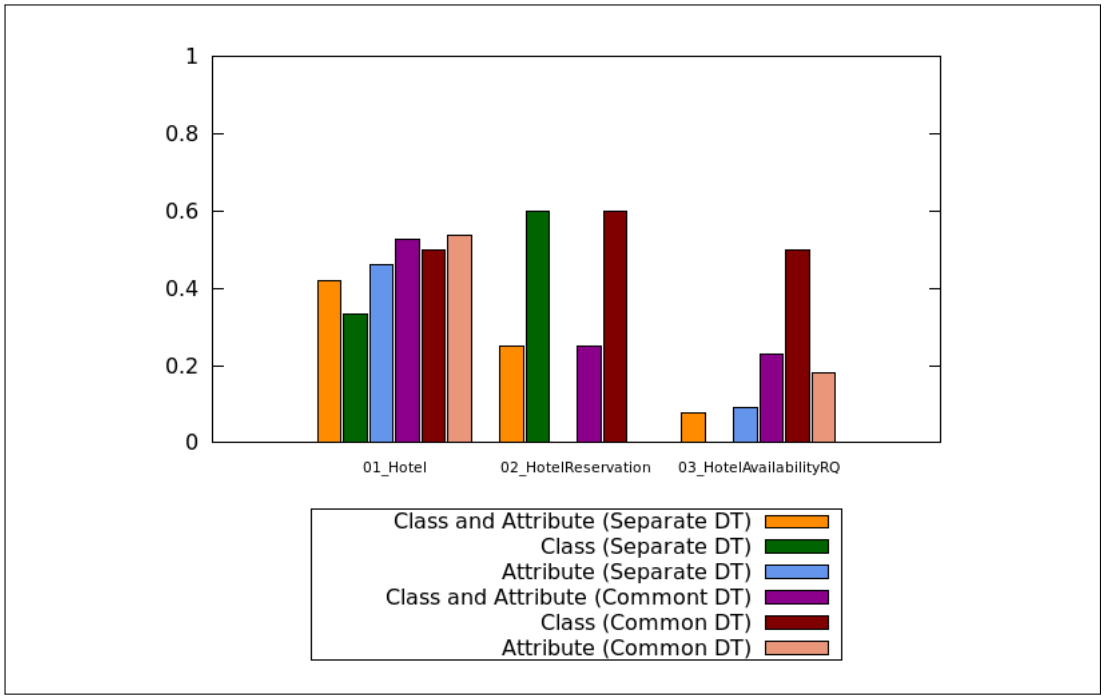


Figure 7.7: Overall for experiment *SeparateTrees*

Post-match effort for adding false negatives and removing false positives is quite high in all cases in Figure 7.7. The highest value of Overall is 0.6.

The hypothesis was not confirmed, all similarity measures are higher for the common decision tree that is trained from a bigger set of training examples, the quality of the decision tree seems to depend more on the size of the set of training samples. This hypothesis is further explored in experiment in Section 7.2.4. The best score was achieved for Precision. Both trees in this experiment did have a larger number of FN than FP. They miss a match suggestion more than they incorrectly suggest it as a match pair. It could be improved by adding an auxiliary source of information or a new matcher.

Examples of matching results from this experiment are shown in Table 7.2. Match pair `numberOfNights - LengthOfStay` is difficult to identify without an auxiliary source of information for both sets of decision tree. Match paires `CheckOutDate - CheckOut`, `ContactInfo - Contact` and `BedType - ReservationType` were identified correctly by common tree and incorrectly by separate decision trees.

7.2.2 Decision Trees Trained with Different Sets of Matchers

The second experiment tries to increase accuracy of matching by expanding the set of available matchers by a new matcher, for example `N-gram`.

	XSD	PIM	DT type	DT	User	Result
C	ContactInfo	Contact	Separate	Mism	Match	FN
C	ContactInfo	Contact	Common	Match	Match	TP
A	Fax	FaxNumber	Separate	Match	Match	TP
A	Fax	FaxNumber	Common	Match	Match	TP
A	numberOfNights	LengthOfStay	Separate	Mism	Match	FN
A	numberOfNights	LengthOfStay	Common	Mism	Match	FN
A	CheckOutDate	CheckOut	Separate	Mism	Match	FN
A	CheckOutDate	CheckOut	Common	Match	Match	TP
A	BedType	ReservationType	Separate	Match	Mism	FP
A	BedType	ReservationType	Common	Mism	Mism	TN

Table 7.2: Examples of mapping results for experiment *SeparateTrees*

```
Experiment Setup:
  Decision Trees Trained with Different Set of Matchers
  (AdditionalMatcher)

  eXolutio Project:
   Travelling
  XSD schemas:
    01_Hotel
    02_HotelReservation
    03_HotelAvailabilityRQ
  Decision tree:
    1. Decision tree trained from the first set of matchers
    (in Figure 7.3)
    2. Decision tree trained from the second set of matchers
    (in Figure 7.8)
  Decision tree training set:
    Set of XSD Schemas: OTA
    Sample count: 55 815 match pairs
  Thesaurus for Dictionary:
    None
  Thesaurus for Matched Thesauri:
    None

  The first Set of Matchers:
    Children
    DataType
    Dictionary
    Length Ratio
    Levenshtein Distance
    Matched Thesauri
    Prefix
  The second Set of Matchers:
```

```
Children
DataType
Dictionary
Length Ratio
Levenshtein Distance
Matched Thesauri
N-gram
Prefix
```

In the second experiment the same set of training samples was used. The common tree for classes and attributes from the first experiment was reused and a new tree was trained with the expanded set of available matchers.

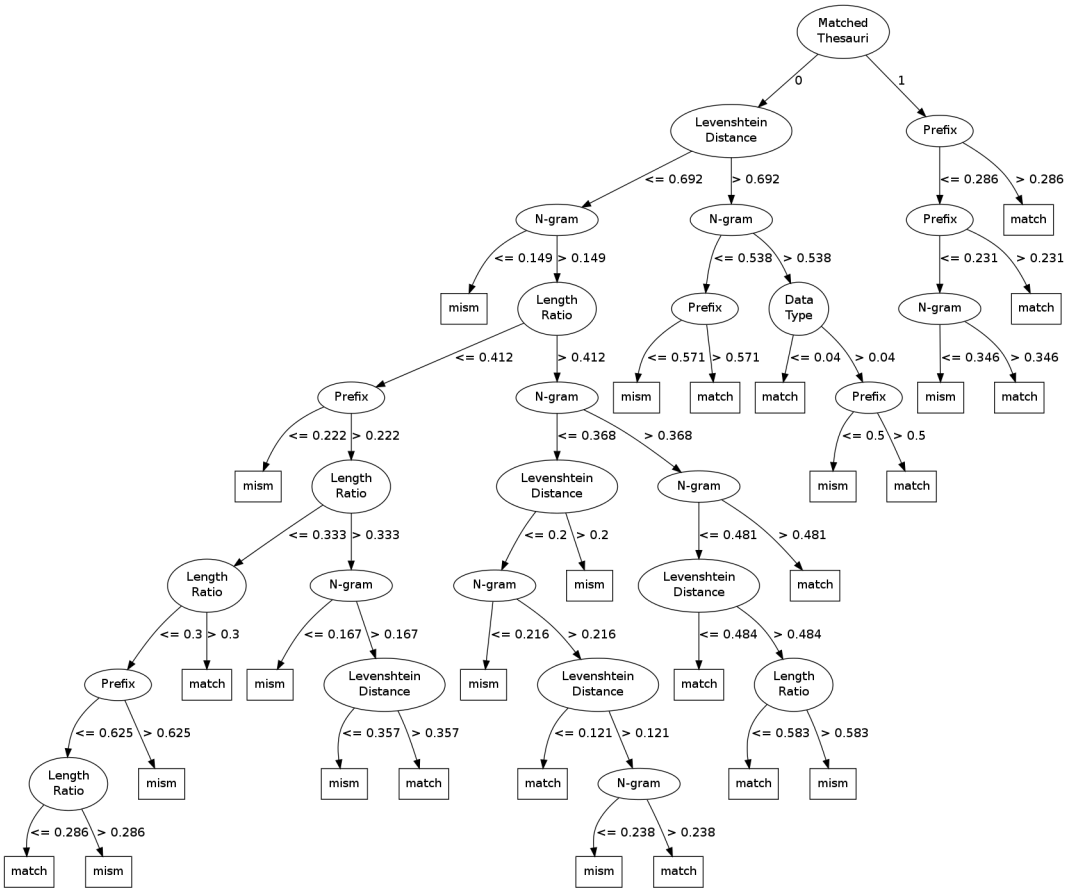


Figure 7.8: Decision tree with N-gram matcher

In Figure 7.8 the new decision tree is shown. It contains matcher `Matched Thesauri` as a root with a subtree for value 1. New matcher `N-gram` is contained in most of the branches and is a parent of most of the leaves. As we can see, this tree is more balanced than the one in Figure 7.3.

Figure 7.9 shows comparison of match quality measures for both decision trees. All values are higher for the new decision tree. Adding a new matcher seems to increase the quality considerably. It lowered both numbers of FP and FN samples. There is a significant improvement of Overall value for schema

`03_HotelAvailability`. It is almost two times higher. Also Precision is slightly increased and no match quality value is lower.

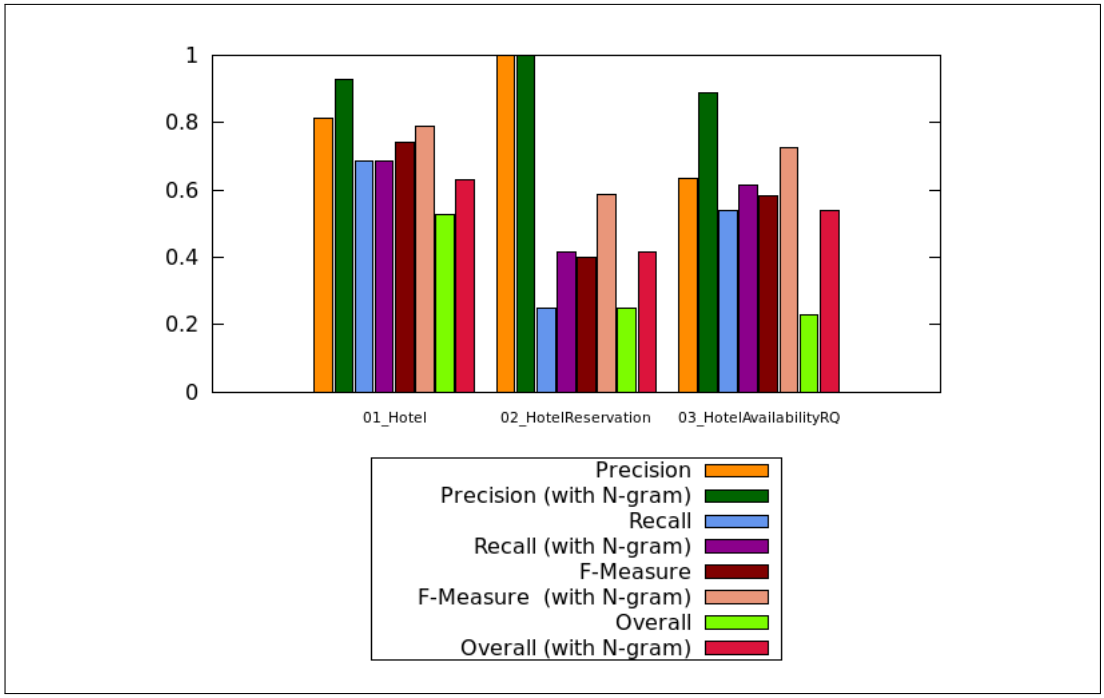


Figure 7.9: Comparison of match quality measures for experiment *Additional-Matcher*

Examples of match pairs which decision trees decided differently are displayed in Table 7.3. Both trees suggest some match pairs incorrectly. `N-gram` tree seems to fail to identify matches that have shorter element names - it suggested as mismatch pairs `CityCode - City` and `Fax - FaxNumber`, that have been correctly identified by the first tree. Both pairs have quite a low value for `N-gram` matcher. On the other hand, the tree without `N-gram` matcher did not suggest as matches pairs `CheckInDate - CheckIn` and `AlternateCurrencyCode - Currency`.

XSD	PIM	DT type	DT	User	Result
CityCode	City	Common	Match	Match	TP
CityCode	City	N-gram	Mism	Match	FN
CheckInDate	CheckIn	Common	Mism	Match	FN
CheckInDate	CheckIn	N-gram	Match	Match	TP
Fax	FaxNumber	Common	Match	Match	TP
Fax	FaxNumber	N-gram	Mism	Match	FN
NumberOfRooms	RoomNumber	Common	Mism	Match	FP
NumberOfRooms	RoomNumber	N-gram	Match	Match	TP
AlternateCurrencyCode	Currency	Common	Mism	Match	FN
AlternateCurrencyCode	Currency	N-gram	Match	Match	TP
Name	StreetName	Common	Match	Mism	FP
Name	StreetName	N-gram	Match	Mism	FP

Table 7.3: Examples of mapping results for experiment *AdditionalMatcher*

Addition of a new matcher seems to have a positive effect in all the aspects of matching. In the following experiment we will explore the possibility of further improvement by adding more auxiliary information.

7.2.3 Usage of Auxiliary Information

In this experiment we compare the quality of match decision as a function of the size and quality of domain thesaurus and the previous user-confirmed matches.

```
Experiment Setup:
  Usage of Auxiliary Information (AuxiliaryInformation)

  eXolutio Project:
   Travelling
  XSD schemas:
    01_Hotel
    02_HotelReservation
    03_HotelAvailabilityRQ
  Decision tree:
    1. ngram (in Figure 7.8)
    2. dictionary (in Figure 7.10)
  Decision tree training set:
    Set of XSD Schemas: OTA
    Sample count: 55 815 match pairs
  Thesaurus for Dictionary:
    Hotel (in Table 7.1)
  Thesaurus for Matched Thesauri:
    Match pairs from previous experiments (in Table B.1)
  Set of Matchers:
    Children
    DataType
    Dictionary
    Length Ratio
    Levenshtein Distance
    Matched Thesauri
    N-gram
    Prefix
```

The tree with `Dictionary` is displayed in Figure 7.10. It is quite similiar to the tree with `N-gram` in Figure 7.8. It contains two subtrees that are exactly the same (green-colored) and two subtrees that have the same matchers but different threshold values (yellow-colored) and one subtree that differs (blue-colored).

In the following Figures 7.11, 7.12, 7.13, 7.14 values of Precision, Recall, F-Measure and Overall are shown. Using the auxiliary information slightly increases Precision for all the schemas.

Figure 7.10: Decision tree with `Dictionary` matcher

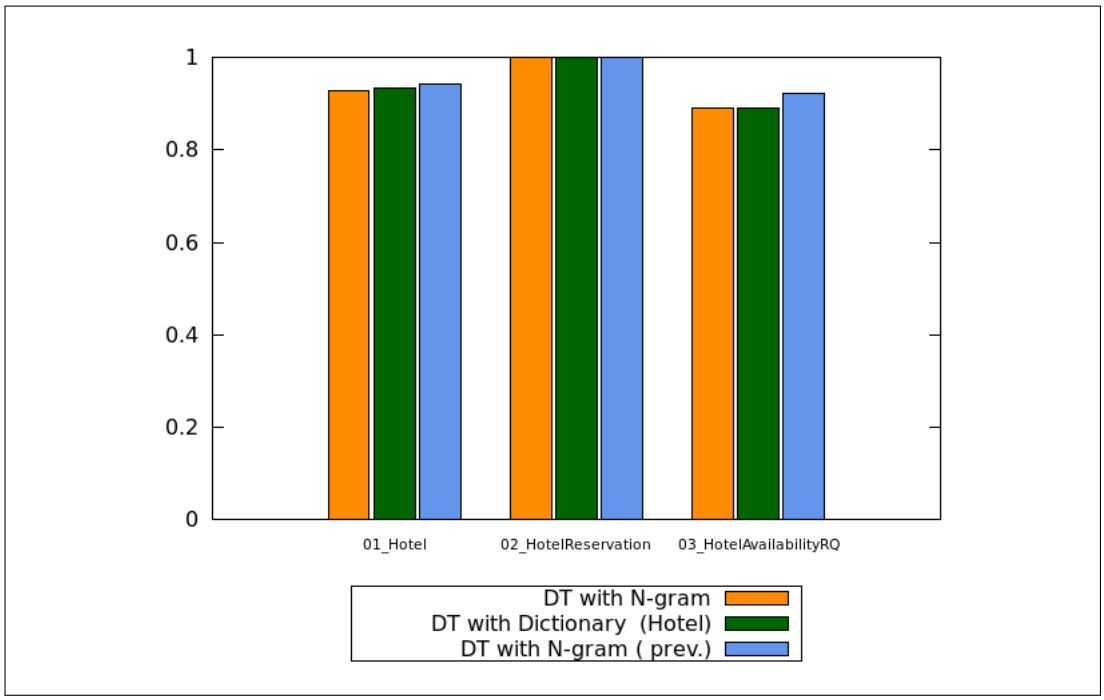


Figure 7.11: Precision for experiment *AuxiliaryInformation*

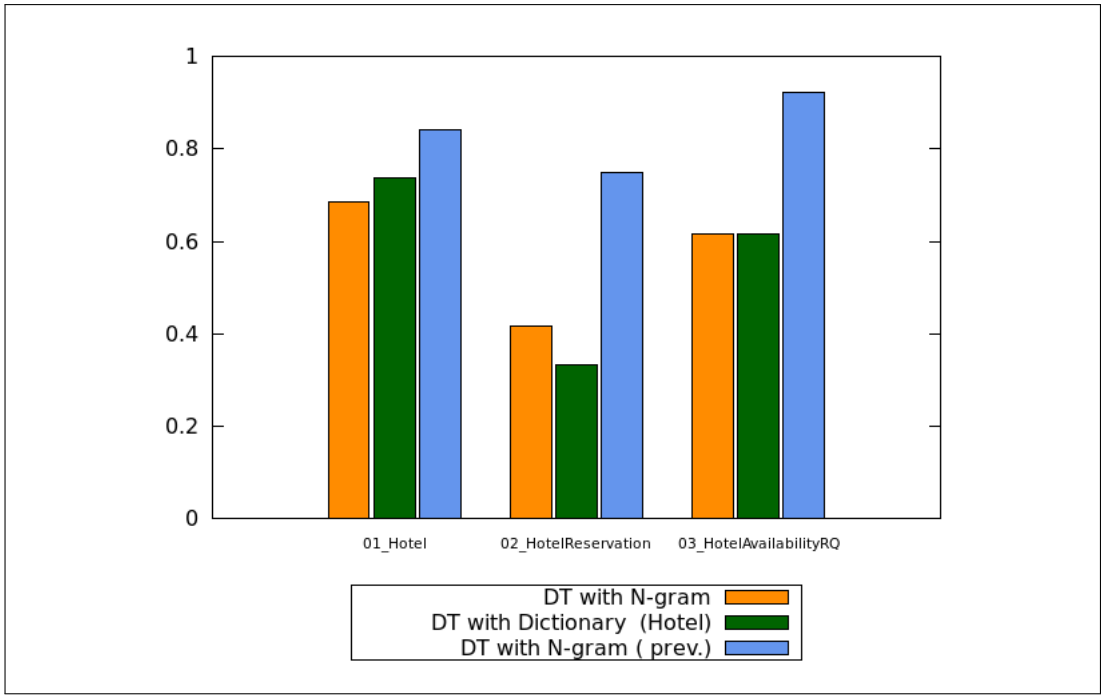


Figure 7.12: Recall for experiment *AuxiliaryInformation*

As we can observe, using the previous matches improves Recall significantly for all the schemas. Domain thesaurus improves value of Recall for the first schema, but it is the same for the third one and lower for the second one. The amount of improvement is less than the setback. The same applies to F-Measure and Overall.

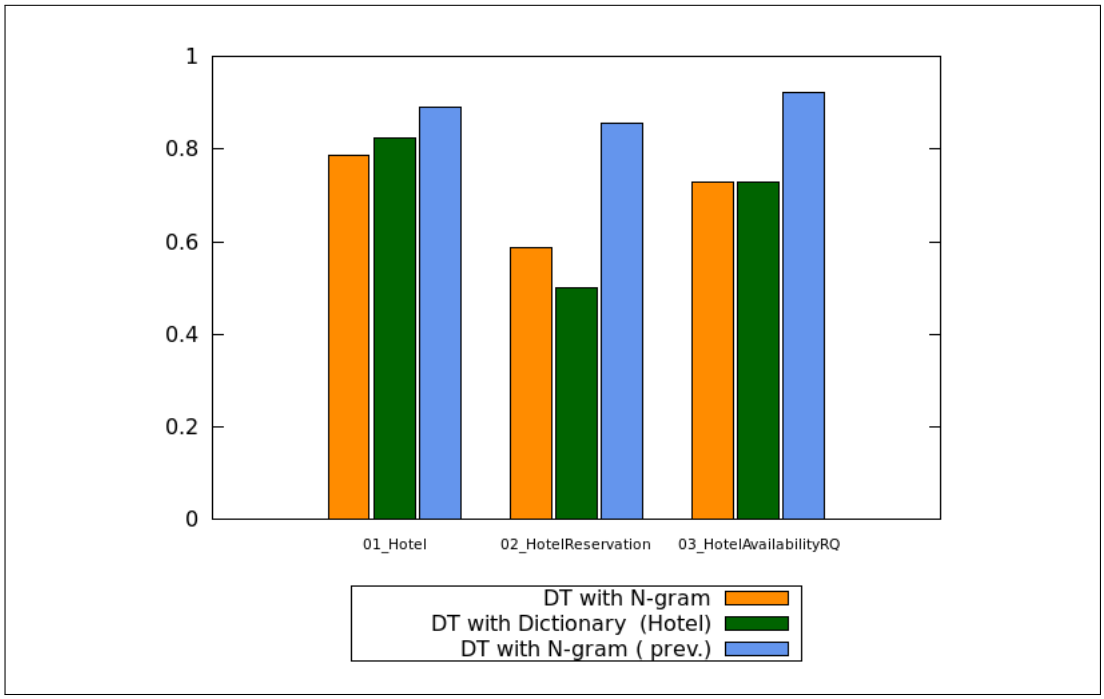


Figure 7.13: F-Measure for experiment *AuxiliaryInformation*

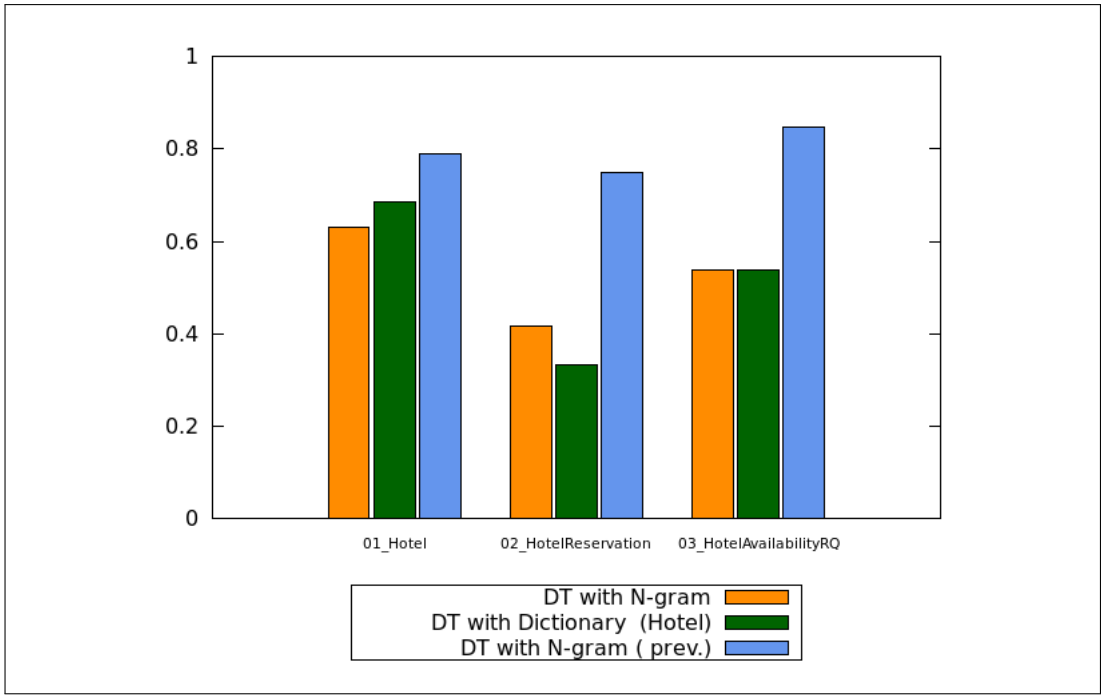


Figure 7.14: Overall for experiment *AuxiliaryInformation*

Table 7.4 shows interesting differences in decision trees output. Experiments that used auxiliary source of information were able to correctly identify match pairs `Location - Destination` and `Location - Address`. Decision tree with `Dictionary` matcher failed to identify the similarity between `ContactInfo - Contact` and `HostelReservation - Reservation`.

	XSD	PIM	DT type	DT	User	
C	ContactInfo	Contact	N-gram	Match	Match	TP
C	ContactInfo	Contact	Dictionary	Mism	Match	FN
C	ContactInfo	Contact	N-gram (prev)	Match	Match	TP
C	Location	Destination	N-gram	Mism	Match	FN
C	Location	Destination	Dictionary	Match	Match	TP
C	Location	Destination	N-gram (prev)	Mism	Match	FN
C	Location	Address	N-gram	Mism	Match	FN
C	Location	Address	Dictionary	Match	Match	TP
C	Location	Address	N-gram (prev)	Match	Match	TP
C	HostelReservation	Reservation	N-gram	Match	Match	TP
C	HostelReservation	Reservation	Dictionary	Mism	Match	FN
C	HostelReservation	Reservation	N-gram (prev)	Match	Match	TP
A	Street	StreetName	N-gram	Mism	Match	FN
A	Street	StreetName	Dictionary	Mism	Match	FN
A	Street	StreetName	N-gram (prev)	Match	Match	TP

Table 7.4: Examples of mapping results for experiment *AuxiliaryInformation*

The `Dictionary` matcher does not improve matching as we assumed, it could be due to the small number of synonyms in the domain thesaurus or a small amount of synonym pairs in compared schemas.

7.2.4 Decision Trees Trained from Training Sample Sets of Different Sizes

The last experiment compares the quality of matching for decision trees trained from sets with various number of training samples.

1. Large set of training data (55 815 match pairs).
2. Smaller set of training data (13 456 match pairs).
3. Small set of training data (4 775 match pairs).

```
Experiment Setup:
  Decision Tree Trained from Training Sample Sets of Different
  Sizes (TrainingSets)

  eXolutio Project:
   Travelling
  XSD schemas:
    01_Hotel
    02_HotelReservation
    03_HotelAvailabilityRQ
  Decision tree:
    1. ngram (in Figure 7.8)
    2. openTrans (in Figure 7.16)
    3. bmeCat (in Figure 7.15)
  Decision tree training set:
    1. Set of XSD Schemas: OTA
       Sample count: 55 815 match pairs
    2. Set of XSD Schemas: OpenTrans
       Sample count: 13 456 match pairs
    3. Set of XSD Schemas: BmeCat
       Sample count: 4 775 match pairs
  Thesaurus for Dictionary:
    None
  Thesaurus for Matched Thesauri:
    None
  Set of Matchers:
    Children
    DataType
    Dictionary
    Length Ratio
    Levenshtein Distance
    Matched Thesauri
    N-gram
    Prefix
```

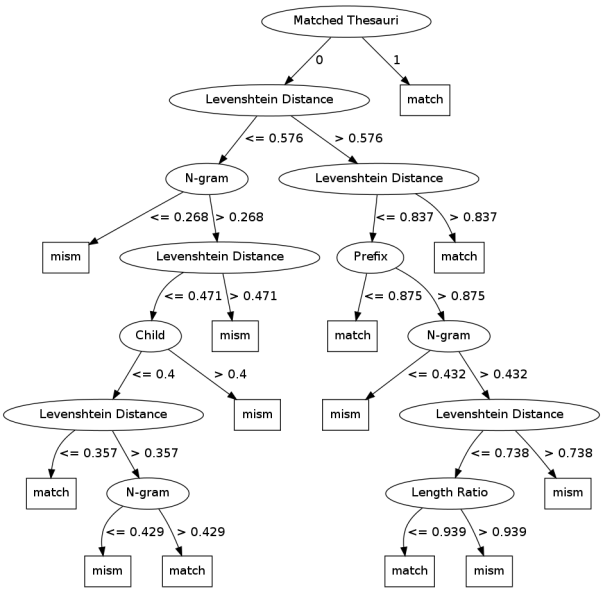


Figure 7.15: Decision tree BmeCat

In Figure 7.15 a decision tree trained from a small set of training samples (4 775 match pairs) is displayed. The training set is prepared from set of schemas `bmecat` – in particular schema pair `bmecat_price - bmecat_product`. It does not contain matchers `DataType` and `Children` and is relatively simple. Root `Matched Thesauri` directly suggests matches for mapping pairs that are contained in the thesaurus of the previous matches. It could be useful if the right thesaurus was used.

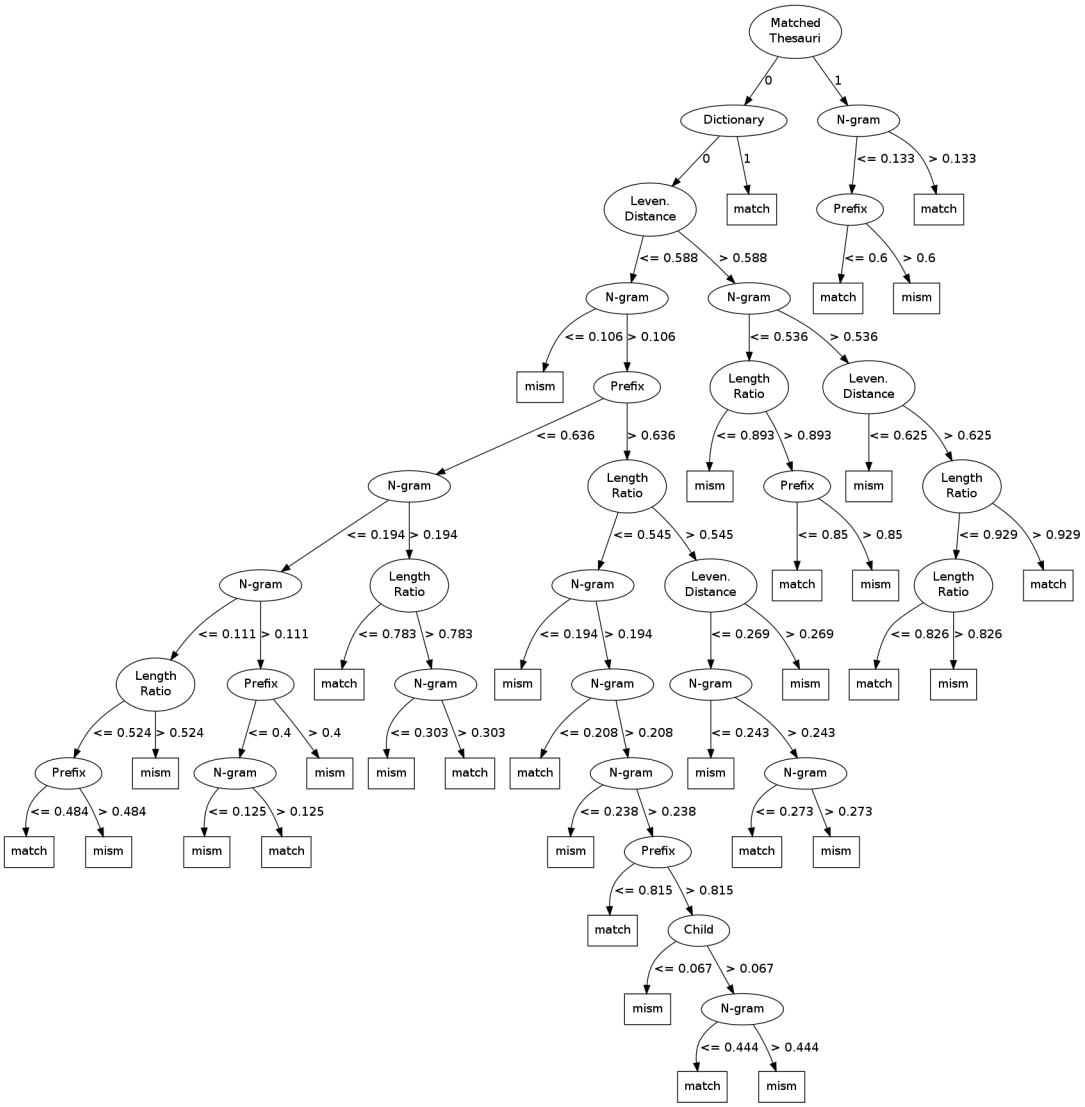


Figure 7.16: Decision tree OpenTrans

Figure 7.16 shows the decision tree that is trained from the set of schemas `openTrans` – in particular schema pair `openbase_1_0 - openTRANS_DISPATCH-NOTIFICATION_1_0`. The training set has 13 456 match pairs. This decision tree contains `Dictionary` matcher even if no domain thesauri was used during training. It is the only decision tree that uses `Children` matcher.

Match quality measures for this experiment are displayed in the following Figures 7.17, 7.18, 7.19 and 7.20.

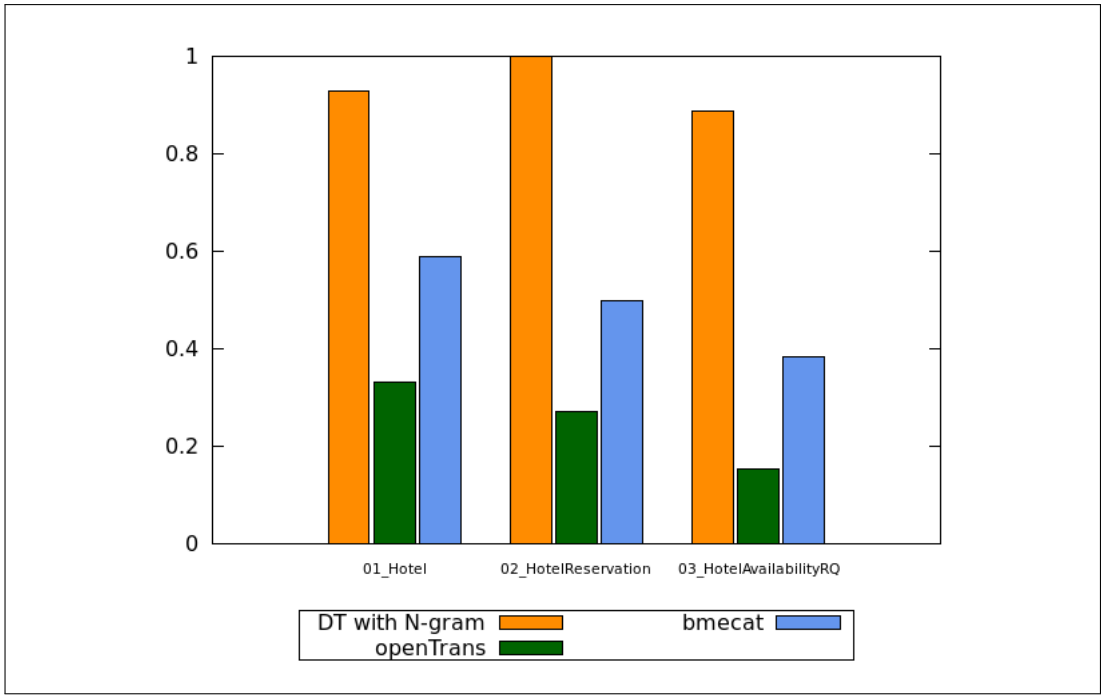


Figure 7.17: Precision for experiment *TrainingSets*

Match quality does not seem to be linear in the size of the training samples. Precision is higher for the decision tree trained from the smallest set of samples than for the second one.

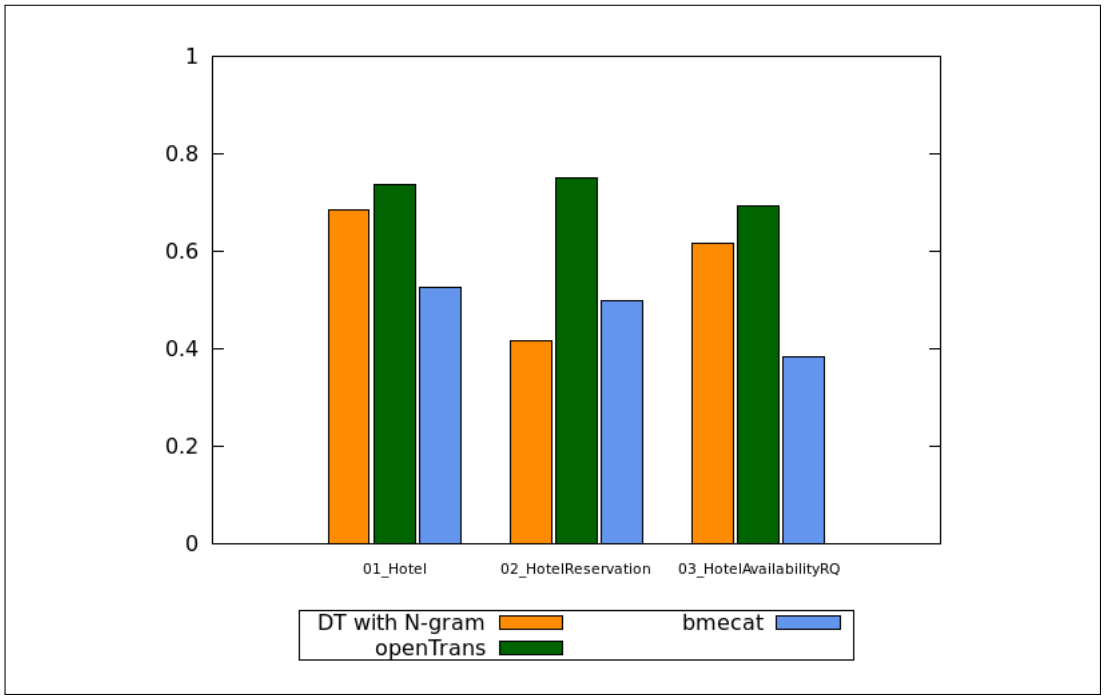


Figure 7.18: Recall for experiment *TrainingSets*

Recall for the second tree is even higher than for the first one. It could be caused by the high number of FP that the second tree identified – total amount

of false positives for all schemas for decision trees is 2, 101 and 21. It is also the reason why the values of Overall are so low – the post effort for removing false positives will be significant.

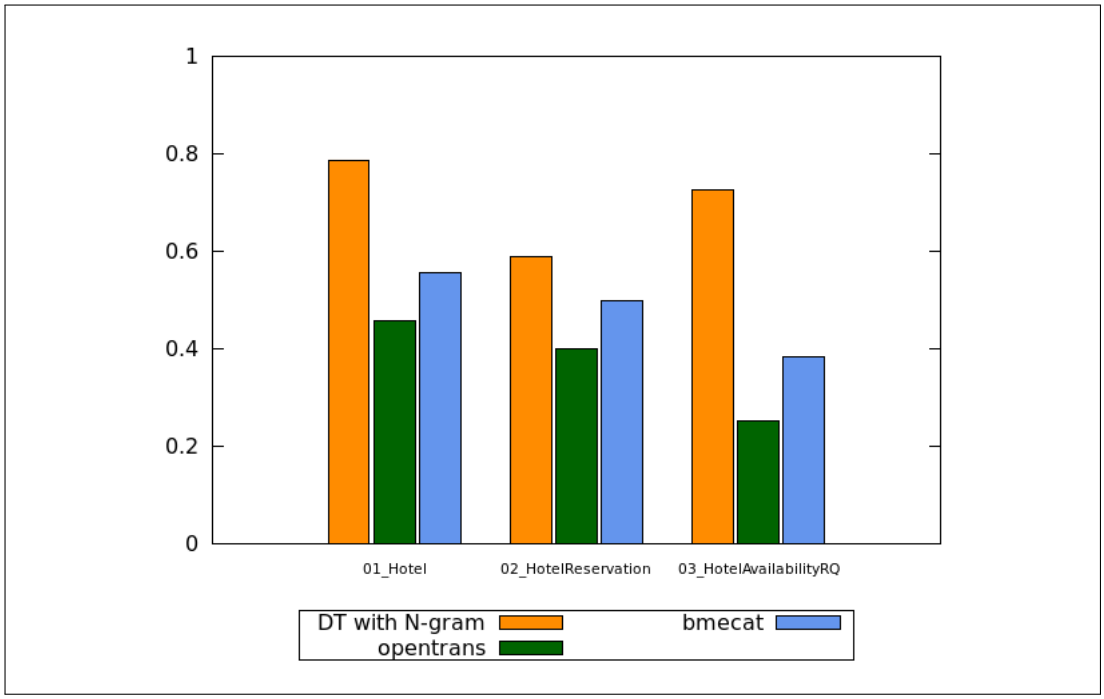


Figure 7.19: F-Measure for experiment *TrainingSets*

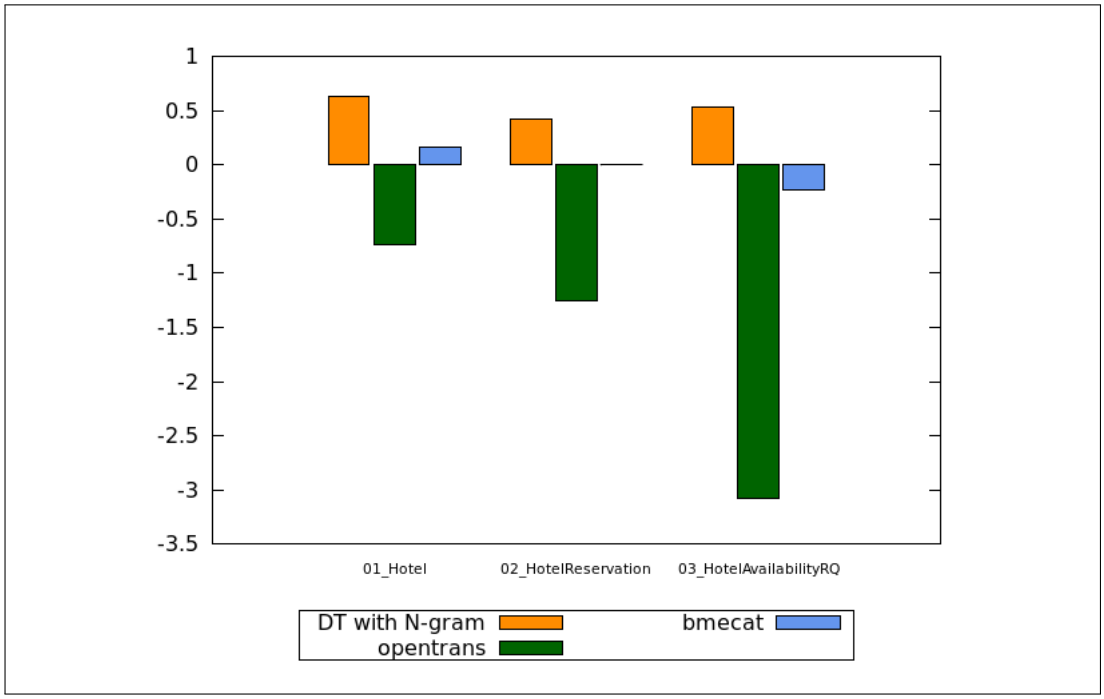


Figure 7.20: Overall for experiment *TrainingSets*

Table 7.5 displays examples of matching results from this expreriment. The new decision trees were able to correctly identify match pair `numberOfGuests`

- `numberOfPerson`, on the other hand they also falsely identified as a match `numberOfGuests - numberOfCourses`.

	XSD	PIM	DT type	DT	User	
C	Hotel	HotelStay	N-gram	Match	Match	TP
C	Hotel	HotelStay	bmecat	Mism	Match	FN
C	Hotel	HotelStay	openTrans	Match	Match	TP
A	Fax	FaxNumber	N-gram	Mism	Match	FN
A	Fax	FaxNumber	bmecat	Mism	Match	FN
A	Fax	FaxNumber	openTrans	Match	Match	TP
A	Street	StreetNumber	N-gram	Match	Match	TP
A	Street	StreetNumber	bmecat	Mism	Match	FN
A	Street	StreetNumber	openTrans	Match	Match	TP
A	CityCode	City	N-gram	Mism	Match	FN
A	CityCode	City	bmecat	Mism	Match	FN
A	CityCode	City	openTrans	Match	Match	TP
A	numberOfGuests	NumberOfPerson	N-gram	Mism	Match	FN
A	numberOfGuests	NumberOfPerson	bmecat	Match	Match	TP
A	numberOfGuests	NumberOfPerson	openTrans	Match	Match	TP
A	numberOfGuests	NumberOfCourses	N-gram	Mism	Mism	TN
A	numberOfGuests	NumberOfCourses	bmecat	Match	Mism	FP
A	numberOfGuests	NumberOfCourses	openTrans	Match	Mism	FP
A	Fax	Tax	N-gram	Mism	Mism	TN
A	Fax	Tax	bmecat	Match	Mism	FP
A	Fax	Tax	openTrans	Match	Mism	FP

Table 7.5: Examples of mapping results for experiment *TrainingSets*

The best results have been achieved with `N-gram` tree with usage of the previous matches, but it still leaves some space for improvement that are suggested in Section 7.2.4. Suprisingly, the `Dictionary` matcher does not improve matching as we assumed. The most promising variation seems to be usage of the previous matches that is well supported in the module – there is possibility for automatic saving of all the user-confirmed matches. The worst results have been delivered by the decision tree OpenTrans – it identified a lot of matches falsely. This tree has been trained from the training set of a medium size.

Conclusion

Schema matching, the problem of finding correspondences, relations or mappings between elements of two schemas, has been extensively researched and has a lot of different applications. In this thesis a particular application of schema matching in MDA (Model-Driven Architecture) is explored. We have implemented our approach within an existing tool called eXolutio that is used for conceptual modeling at two levels of MDA – platform independent and platform specific. In this architecture there exist mappings between elements of schemas at these two levels, called interpretation of a PSM schema against a PIM schema. This mapping is very useful in case of a change, because changes in one place are propagated to all the related schemas. Our schema matching approach is used to identify these mappings – the PSM element - PIM element pair is identified as an interpretation of PSM element against PIM element if it is suggested as a match by schema matching algorithm.

We explored various approaches to schema matching and selected the most promising possible approach for our application – schema matching using a decision tree. This solution is dynamic, versatile, highly extensible, quick and has a low level of user intervention and a low level of required auxiliary information. We extended the previous work of Jakub Stárka [22]. In particular, we utilized C.50 algorithm for training of decison tree from a large set of user-annotated schema pairs. Our approach is more versatile, extensible and reusable. Further we evaluated our approach on a wide range of experiments and implemented a module that is easily extensible.

In particular, we make the following contributions:

- We created an implementation with an easily extensible set of individual matchers and match quality measures.
- We implemented a user-friendly interface for evaluation of mappings suggested by the decision tree.
- We provided a set of user-annotated training samples that could be reused in further experiments.
- We proposed several variations that could improve the accuracy of matching. The following experiments were performed. We trained separate trees for classes and attributes and compared the quality of the mapping results with the results of the common tree. Then we extended the set of matchers and used auxiliary source of information. Furthermore, we compared quality of matching in dependence on the size of training set.
- We evaluated the proposed variations using automatic match quality measures – Precision, Recall, F-Measure and Overall.
- We explored results of experiments that evaluated the proposed variations of decision tree training and identified the best variation – decision tree with matcher `N-gram` with usage of the previous matches.

- We provided a solid background for further experiments.

Future Work

A straightforward extension of this work is to expand the set of available matchers with more powerfull matchers, e.g. with a matcher that uses the WordNet[7] thesaurus for synonyms. Further possibilities are for example string matchers [8] and Soundex matcher.

Also the user interface leaves a space for improvement. We could add an interface for evaluation of matches during the preparation phase of decision tree training or dynamic editation of trained decision tree – remove, move, add matcher node, change results in leaves or threshold on edges.

[7] `http://wordnet.princeton.edu/`
[8] `http://secondstring.sourceforge.net/`

Bibliography

[1] Do, H. H., Rahm, E.: *COMA - A system for flexible combination of schema matching approaches*. Proceedings of the 28th international conference on Very Large Data Bases, Pages 610-621, VLDB Endowment, Hong Kong, China, 2002.

[2] Duchateau, F., Bellahsene, Z., Coletta, R.: *A Flexible Approach for Planning Schema Matching Algorithms*. On the Move to Meaningful Internet Systems: OTM 2008, Pages 249-264, Springer Berlin Heidelberg, 2008 ISBN: 978-3-540-88870-3.

[3] Rahm, E., Bernstein, P. A.: *A survey of approaches to automatic schema matching*. The VLDB Journal – The International Journal on Very Large Data Bases Volume 10 Issue 4, Pages 334-350, Springer-Verlag New York, Inc. Secaucus, NJ, USA, December 2001, ISSN: 1066-8888.

[4] Madhavan, J., Bernstein, P. A., Rahm, E.: *Generic Schema Matching with Cupid*. Proceeding VLDB '01 Proceedings of the 27th International Conference on Very Large Data Bases, Pages 49-58, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2001, ISBN:1-55860-804-4.

[5] Melnik, S., Garcia-Molina, H., Rahm, E.: *Similarity Flooding: A Versatile Graph Matching Algorithm*. Proceeding ICDE '02 Proceedings of the 18th International Conference on Data Engineering, Page 117, IEEE Computer Society Washington, DC, USA, 2002.

[6] Do, H. H., Melnik, S., Rahm, E.: *Comparison of Schema Matching Evaluations*. Proceeding Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, Pages 221-237, Springer-Verlag London, UK, 2003, ISBN:3-540-00745-8.

[7] Doan, A., Domingos, P., Halevy, A.: *Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach*. Proceeding SIGMOD '01 Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Pages 509-520, ACM New York, NY, USA, 2001, ISBN:1-58113-332-4.

[8] Doan, A., Madhavan, J., Domingos, P., Halevy, A.: *Learning to Map between Ontologies on the Semantic Web*. Proceeding WWW '02 Proceedings of the 11th international conference on World Wide Web, Pages 662-673, ACM New York, NY, USA, 2002, ISBN:1-58113-449-5.

[9] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F.: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation, 26 November 2008, `http://www.w3.org/TR/REC-xml`.

[10] Thompson, H. S., Beech, D., Maloney, M., Mendelsohn, N.: *XML Schema Part 1: Structures (Second Edition)*. W3C Recommendation, 28 October 2004, `http://www.w3.org/TR/xmlschema-1/`.

[11] Clark, J., Murata, M.: *RELAX NG Specification*. Committee Specification, 3 December 2001, `http://relaxng.org/spec-20011203.html`.

[12] ISO/IEC 19757-3:2006. *Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron*. International Organization for Standardization and International Electrotechnical Commission, 2006.

[13] *Gnuplot, an Interactive Plotting Program*, `http://www.gnuplot.info/`.

[14] *Graphviz, Graph Visualization Software*, `http://www.graphviz.org/`.

[15] Aumueller, D., Do, H. H., Massmann, S., Rahm, E.: *Schema and Ontology Matching with COMA++*. Proceeding SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data, Pages 906-908, ACM New York, NY, USA 2005, ISBN:1-59593-060-4.

[16] Quinlan, J. R.: *Induction of Decision Trees*. Machine Learning Volume 1 Issue 1, Pages 81-106, Kluwer Academic Publishers Hingham, MA, USA, 1986.

[17] Quinlan, J. R.: *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993, ISBN:1-55860-238-0.

[18] Breiman, L., Friedman, J., Olshen, R., Stone, C.: *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[19] Hunt, E. B., Marin, J., Stone, P. T.: *Experiments in Induction*. Academic Press, New York, NY, 1966.

[20] Mehta, M., Agrawal, R., Rissanen, J.: *SLIQ: A fast scalable classier for data mining*. Proceeding EDBT '96 Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology, Pages 18-32, Springer-Verlag London, UK, 1996 ISBN:3-540-61057-X.

[21] *Documents associated with Unified Modeling Language (UML), v2.4.1*. August 2011, `http://www.omg.org/spec/UML/2.4.1/`.

[22] Stárka, J.: *S*imilarity of XML Data. Master's thesis, Charles University in Prague, 2010.

[23] Duchateau, F., Bellahsene, Z., Coletta, R.: *A* Selftuning Approach for Improving Composite Schema Matchers. `http://hal-lirmm.ccsd.cnrs.fr/lirmm-00271534`, 2008

[24] Flesca, S., Manco, G., Masciari, E., Pontieri, L., Pugliese, A.: *Detecting Structural Similarities Between XML Documents*. In Proc. of the 5th Intl. Workshop on the Web and Databases, Pages 55-60, 2002.

[25] Shasha, D., Zhang, K.: *Approximate Tree Pattern Matching*. Pattern Matching Algorithms, Oxford University Press, 1997, Pages 341–371.

[26] Nierman, A., Jagadish, H. V.: *Evaluating Structural Similarity in XML Documents*. Proceedings of the Fifth International Workshop on the Web and Databases, 2002, Pages 61–66.

[27] Li, W., Clifton, C.: *SemInt: a tool for identifying attribute correspondences in heterogeneous databases using neural network*. Data & Knowledge Engineering, Volume 33, Issue 1, April 2000, Pages 49-84, ISSN 0169-023X.

[28] Chen, P.: *The Entity-Relationship Model – Toward a Unified View of Data*. ACM Transactions on Database Systems, Pages 9-36, Mar. 1976.

[29] Quinlan, R.: *C5.0*. `http://www.rulequest.com/see5-unix.html`.

[30] Stárka, J., Mlýnková, I., Klímek, J., Nečaský, M.: *Integration of web service interfaces via decision trees*. in Proceedings of the 7th International Symposium on Innovations in Information Technology, Abu Dhabi, IEEE Computer Society, 2011, Pages 47-52, ISBN: 978-1-4577-0311-9.

[31] Gross, A., Hartung, M., Kirsten, T., Rahm, E.: *GOMMA Results for OAEI 2012*. Seventh International Workshop on Ontology Matching ISWC, 2012.

[32] Jiménez-Ruiz, E., Cuenca Grau, B.: *LogMap: Logic-based and Scalable Ontology Matching*. Proceeding ISWC'11 Proceedings of the 10th international conference on The semantic web - Volume Part I, Springer-Verlag Berlin, Heidelberg, 2011, Pages 273-288, ISBN: 978-3-642-25072-9.

[33] He, H., Meng, W., Yu, C., Wu, Z.: *Automatic integration of Web search interfaces with WISE-Integrator*. The VLDB Journal September 2004, Volume 13, Issue 3, Springer-Verlag, 2004, Pages 256-273, ISSN: 1066-8888.

[34] Jaewook, K., Peng, Y., Ivezic, N., Shin, J.: *An Optimization Approach for Semantic-based XML Schema Matching*. International Journal of Trade, Economics, and Finance, January 2011, Pages 78-86.

[35] Do, H. H., Rahm, E.: *Flexible Integration of Molecular-biological Annotation Data: The GenMapper Approach*. 9th International Conference on Extending Database Technology, Springer Berlin Heidelberg, 2004, Pages 811-822, ISBN: 978-3-540-21200-3.

[36] Goble, C., Stevens, R., Ng, G., Bechhofer, S., Paton, N. W., Baker, P. G., Peim, M., Brass, A.: *Transparent Access to Multiple Bioinformatics Information Sources*. Oxford Journals, Life Sciences & Mathematics & Physical Sciences, Bioinformatics, Volume 16, Issue 2, 1999, Pages 184-186.

[37] Kirsten, T., Do, H. H., Körner, Ch., Rahm, E.: *Hybrid Integration of molecular-biological Annotation Data*. Proc. 2nd International Workshop on Data Integration in the Life Sciences (DILS), San Diego, Springer Berlin Heidelberg, 2005 Pages 208-223, ISBN 978-3-540-27967-9.

[38] Calvanese, D., Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: *Data Integration and Reconciliation in data Warehousing: Conceptual Modeling and Reasoning Support*. Networking and Information Systems, 1999, Pages 413-432.

[39] Thor, A., Rahm, E.: *CloudFuice: A flexible Cloud-based Data Integration System*. Web Engineering Lecture Notes in Computer Science Volume 6757, Springer Berlin Heidelberg, 2011, Pages 304-318, ISBN: 978-3-642-22232-0

[40] Wojnar, A.: *Similarity of XML Data*. Master's thesis, Charles University in Prague, 2008.

[41] Klímek, J., Mlýnková, I. Nečaský, M.: *eXolutio: Tool for XML and Data Management*. In CEUR Workshop Proceedings, 2012, Pages 69-80, ISSN: 1613-0073.

[42] Nikovski, D., Esenther, A., Ye, X., Shiba, M., Takayama, S.: *Bayesian Networks for Matcher Composition in Automatic Schema Matching*. ICEIS 1, SciTePress, 2012 Pages 48-55, ISBN: 978-989-8565-10-5.

[43] Larson, J. A., Navathe, S. B., Elmasri, R.: *A theory of attribute equivalence in databases with application to schema integration*. Software Engineering, IEEE Transactions on (Volume: 15, Issue: 4), 1989, Pages 449-463, ISSN: 0098-5589.

[44] *The Soundex Indexing System*. `http://www.archives.gov/research/census/soundex.html`

[45] Biron, P. V., Permanente, K., Malhotra, A.: *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation, 28 October 2004, `http://www.w3.org/TR/xmlschema-2/`

[46] Cheng, W., Sun, Y.: *GSMA: A Structural Matching Algorithm for Schema Matching in Data Warehousing*. Fuzzy Systems and Knowledge Discovery, Lecture Notes in Computer Science Volume 3614, Springer Berlin Heidelberg, 2005, Pages 408-411, ISBN: 978-3-540-28331-7.

[47] Hong-Minh, T., Smith, D.: *Hierarchical Approach for Datatype Matching in XML Schemas*. BNCOD '07. 24th British National Conference on Databases, 2007, Pages 120-129, ISBN: 0-7695-2912-7.

[48] Bergamashchi, S., Castano, M., Vincini, M: *Semantic Integration of Semistructured and Structured Data Sources*. ACM SIGMOD Record, Volume 28 Issue 1, ACM New York, NY, USA, 1999, Pages 54-59.

[49] Fensel, D., Horrocks, I., Harmelen, F., McGuinness, D., Patel-Schneider, P.: *OIL: Ontology Infrastructure to Enable the Semantic Web*. IEEE Intelligent Systems, Volume 16, 2001, Pages 200-201.

[50] Candan, K. S., Liu, H., Suvarna, R.: *Resource description framework: metadata and its applications*. ACM SIGKDD Explorations Newsletter, Volume 3 Issue 1, ACM New York, NY, USA, 2001, Pages 6-19.

[51] Miller, J., Mukerji, J.: *MDA Guide Version 1.0.1.* Object Management Group, 2003. http://www.omg.org/docs/omg/03-06-01.pdf

[52] *OWL Web Ontology Language Overview.* W3C Recommendation, 2004. http://www.w3.org/TR/owl-features/

[53] *SHOE.* Parallel Understanding Systems Group, Department of Computer Science, University of Maryland at College Park http://www.cs.umd.edu/projects/plus/SHOE/

[54] Nečaský, M., Mlýnková, I., Klímek, J., Malý, J.: *When conceptual model meets grammar: A dual approach to XML data modeling.* International Journal on Data & Knowledge Engineering, volume 72, Pages 1-30, Elsevier, February 2012, ISBN:3-642-17615-1 978-3-642-17615-9.

List of Tables

List of Figures

List of Algorithms

A. Appendix: Content of DVD

The DVD is enclosed to this thesis. It contains the text of the work, the source code of the Adaptive Similarity of XML Data module and the source code of the eXolutio tool. Furthermore, it contains all the used data – the sets of training samples, input schemas, trained decision trees and outputs of experiments.

The DVD has the following directory structure:

- **bin** This directory contains installation of the eXolutio tool with the Adaptive Similarity of XML Data module.
- **data** This directory contains all the data used for preparation for training, training samples, trained decision trees, input schemas and domain thesaurus.
- **doc** This directory contains user documentation to the Adaptive Similarity of XML Data module.
- **experiments** This directory contains setting and results of all the experiments that were performed in this work.
- **src** This directory contains the source code of the eXolutio tool and the source code of the Adaptive Similarity of XML Data module.
- **text** This directory contains the text of this thesis in PDF format.

B. Appendix: Data Used in Experiments

In this chapter schemas that are used in experiments are displayed. There is also table with thesaurus for matcher `Matched Thesauri` used in the experiment *AuxiliaryInformation*.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="http://www.example.com/schema/hotel"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.com/schema/hotel"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xs:element name="Hotel">
    <xs:complexType>
      <xs:attribute name="Name" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="ContactInfo">
    <xs:complexType>
      <xs:attribute name="Fax" type="xs:string" />
      <xs:attribute name="PhoneNumber" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Location">
    <xs:complexType>
      <xs:attribute name="City" type="xs:string" />
      <xs:attribute name="Street" type="xs:string" />
      <xs:attribute name="PostalCode" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="RoomDetail">
    <xs:complexType>
      <xs:attribute name="NumberOfAdults"
                    type="xs:int"
                    use="optional" />
      <xs:attribute name="NumberOfRooms"
                    type="xs:int"
                    use="optional"
                    default="1" />
      <xs:attribute name="BedType"
                    type="xs:string"
                    use="optional" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure B.1: XML schema `01_Hotel.xsd`

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns="http://kusad5am.mff.cz/hb/schema/reservation"
           elementFormDefault="qualified"
           targetNamespace="http://kusad5am.mff.cz/hb/
                            schema/reservation"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ReservationRequest"
              type="HostelReservation" />
  <xs:element name="ReservationResponse"
              type="HostelReservation" />
  <xs:complexType name="HostelReservation">
    <xs:sequence>
      <xs:element name="ChosenHostel" type="Hostel"
                  minOccurs="0" />
      <xs:element name="Customer" type="Customer" />
      <xs:element name="PaymentCredentials"
                  type="PaymentCredentials" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="arrivalDate" type="xs:date"
                  use="required" />
    <xs:attribute name="numberOfGuests" type="xs:int"
                  use="required" />
    <xs:attribute name="numberOfNights" type="xs:int"
                  use="required" />
    <xs:attribute name="ReservationStatus" type="xs:string"
                  use="required" />
    <xs:attribute name="id" type="xs:int" use="required" />
  </xs:complexType>
  <xs:complexType name="Hostel">
    <xs:sequence>
      <xs:element name="ChosenRoom" type="RoomType"
                  minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="id" type="xs:int" use="required" />
  </xs:complexType>
  <xs:complexType name="RoomType">
    <xs:attribute name="id" type="xs:string" use="required" />
  </xs:complexType>
  <xs:complexType name="Customer">
    <xs:attribute name="id" type="xs:int" use="required" />
    <xs:attribute name="surname" type="xs:string"
                  use="required" />
    <xs:attribute name="name" type="xs:string"
                  use="required" />
  </xs:complexType>
  <xs:complexType name="PaymentCredentials">
    <xs:attribute name="paymentToken" type="xs:string"
                  use="required" />
  </xs:complexType>
</xs:schema>
```

Figure B.2: XML schema `02_HotelReservation.xsd`

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
  <xsd:element name="HotelAvailabilityRQ">
    <xsd:complexType>
      <xsd:attribute name="ChainCode" type="xsd:string" />
      <xsd:attribute name="PropertyCode" type="xsd:string" />
      <xsd:attribute name="CheckInDate" type="xsd:date" />
      <xsd:attribute name="CheckOutDate" type="xsd:date" />
      <xsd:attribute name="NumberOfNights" type="xsd:integer" />
      <xsd:attribute name="NumberOfPersons"
                     type="xsd:integer" />
      <xsd:attribute name="CityCode" type="xsd:string" />
      <xsd:attribute name="AlternateCurrencyCode"
                     type="xsd:string" />
      <xsd:attribute name="MoreRoomsToken" type="xsd:integer" />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="HotelOptions">
    <xsd:complexType>
      <xsd:attribute name="NumberOfRooms" type="xsd:integer" />
      <xsd:attribute name="RoomType" type="xsd:string" />
      <xsd:attribute name="RateCode" type="xsd:string" />
      <xsd:attribute name="BedType" type="xsd:string" />
      <xsd:attribute name="RateAccess" type="xsd:string" />
      <xsd:attribute name="RateCategory" type="xsd:string" />
      <xsd:attribute name="RateRange" type="xsd:integer" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Figure B.3: XML schema `03_HotelAvailabilityRQ.xsd`

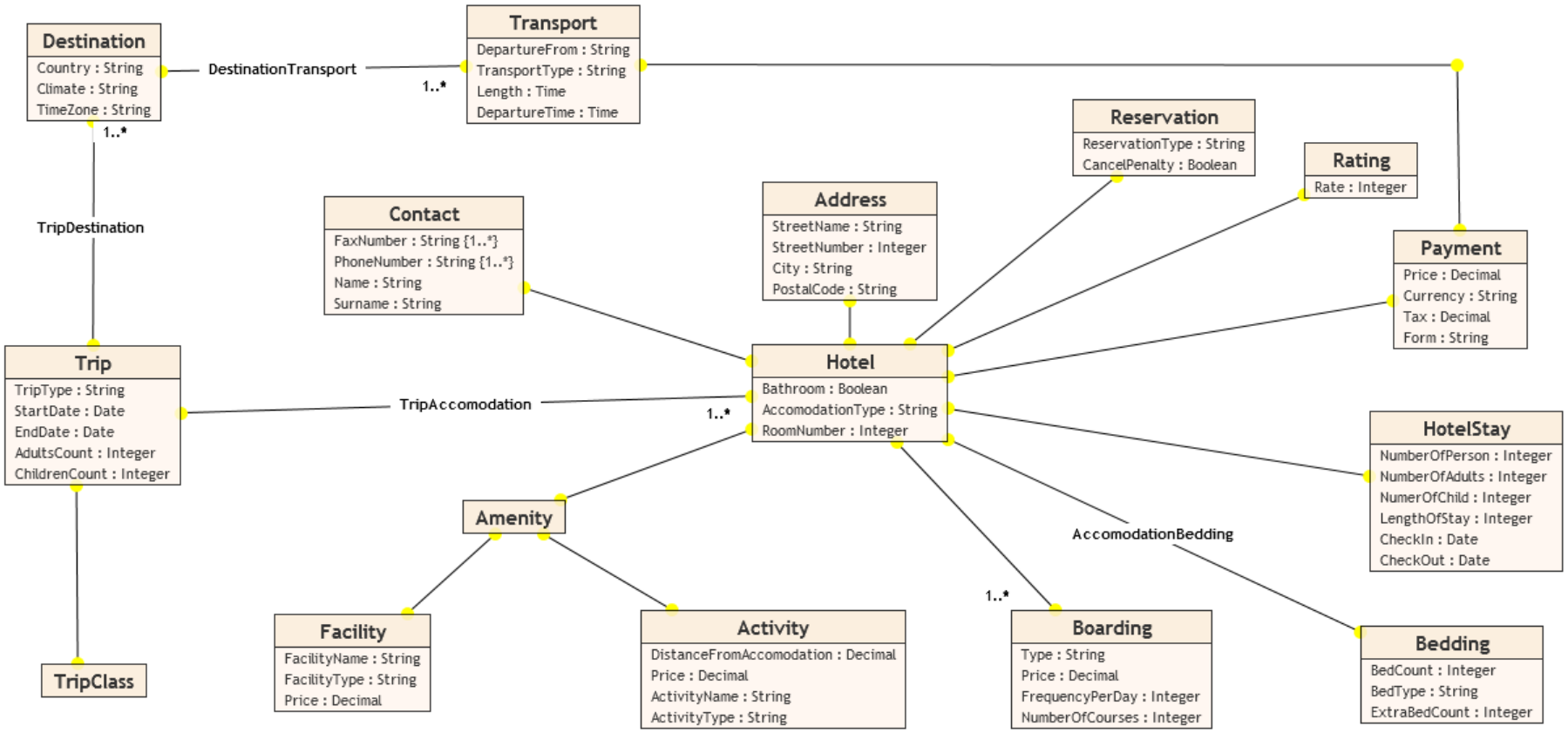


Figure B.4: PIM schema for Holiday Planning

PSM element	PIM element
Hotel	Hotel
Hotel	HotelStay
ContactInfo	Contact
Location	Destination
Location	Address
RoomDetail	Bedding
Name	Name
Name	Surname
Fax	FaxNumber
PhoneNumber	PhoneNumber
City	City
Street	StreetName
Street	StreetNumber
PostalCode	PostalCode
NumberOfAdults	AdultsCount
NumberOfAdults	NumberOfAdults
NumberOfRooms	RoomNumber
BedType	Type
BedType	BedType
HostelReservation	Reservation
Hostel	Hotel
RoomType	Bedding
PaymentCredentials	Payment
ReservationResponse	Reservation
arrivalDate	StartDate
numberOfGuests	NumberOfPerson
numberOfNights	LengthOfStay
surname	Name
surname	Surname
name	Name
name	Surname
HotelAvailabilityRQ	Hotel
HotelOptions	Hotel
CheckInDate	CheckIn
CheckOutDate	CheckOut
NumberOfNights	LengthOfStay
NumberOfPersons	NumberOfPerson
CityCode	City
CityCode	PostalCode
AlternateCurrencyCode	Currency
NumberOfRooms	RoomNumber
RoomType	Type
BedType	Type
BedType	BedType

Table B.1: Previously confirmed matches used in experiment *AuxiliaryInformation*