UNIVERSITÄT
KOBLENZ · LANDAU

Megamodel-driven Traceability Recovery of Correspondence & Conformance Links among O/R/X-Mapping Artifacts

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informatik

vorgelegt von

Maximilian Meffert

Erstgutachter:	Prof. Dr. Ralf Lämmel Institut für Informatik
Zweitgutachter:	Msc. Johannes Härtel Institut für Informatik

Koblenz, im November 2017

Erklärung

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet–Quellen – benutzt habe und die Arbeit von mir vorher nicht in einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium (CD-Rom).

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

....

(Ort, Datum) (Maximilian Meffert)

Zusammenfassung

TBD.

Abstract

TBD.

Acknowledgements

TBD.

Contents

List of Figures

Chapter 1

Introduction

TBD.

Contributions

Non-Contributions

Chapter 2

Related Work

TBD.

Chapter 3

Background

This chapter summarizes the necessary background topics of the thesis. Each topic is introduced as independent as possible, interrelation is done during synthesis of hypotheses for this thesis in chapter 4. However, some of the following sections are sorted in a way that one may be based on its predecessor. In particular sections on the necessary mathematical background share definitions. Introducing them repeatedly would be redundant.

3.1 Predicate Logic

3.2 Relations

This section introduces the necessary aspects of mathematical relations for this thesis. The concept of relations is a generalization of semantic dependencies between two or more mathematical objects. This section is based on [13].

Relations are based on set-theory. We also introduce the necessary constructs of set-theory in order to clarify terminology and notation. A set is a collection of well distinguishable mathematical objects. Objects in a set are called elements of the set. A set does not contain two or more identical elements. The notation $x \in X$ denotes that x is an element of the set X. The symbol $\emptyset$ denotes the *empty set*, which contains no elements. The symbol Ω denotes the universal set, which contains all elements.

Definition 1 (Inclusion) *Let X and Y be a sets. Y* includes *X if and only if:*

$$X \subset Y :\Leftrightarrow \forall x[x \in X \rightarrow x \in Y] \Leftrightarrow \forall x[x \notin X \vee x \in Y] \tag{3.1}$$

Then X is called subset *of Y and Y is called* superset *of X. For an arbitrary set $Z \neq \emptyset$, the statement $\emptyset \subset Z$ is always true, respectively $Z \subset \emptyset$ is always false.*

We also define the opposite property: Y does not include X if and only if:

$$X \not\subset Y :\Leftrightarrow \exists x[x \in X \wedge x \notin Y] \tag{3.2}$$

Definition 2 (Union) *Let X and Y be a sets.*

$$X \cup Y := \{x | x \in X \vee x \in Y\} \tag{3.3}$$

$X \cup Y$ is called union *of X and Y.*

Definition 3 (Intersection) *Let X and Y be a sets.*

$$X \cap Y := \{x | x \in X \wedge x \in Y\} \tag{3.4}$$

$X \cap Y$ is called intersection *of X and Y.*

Definition 4 (Power-Set) *Let X be a set, then the* power-set *of X is defined as:*

$$\mathcal{P}(X) := \{Y | Y \subset X\} \tag{3.5}$$

The definition of inclusion provides an order for power-sets. So we may compare two sets A and B in the sense of *smaller* and *larger*, i.e.:

$$A \text{ is smaller than } B \Leftrightarrow A \subset B \tag{3.6}$$

$$A \text{ is larger than } B \Leftrightarrow B \subset A \tag{3.7}$$

$$A \text{ is the smallest subset of } B \Leftrightarrow \forall C \in \mathcal{P}(B) : A \subset C \tag{3.8}$$

$$A \text{ is the largest subset of } B \Leftrightarrow \forall C \in \mathcal{P}(B) : C \subset A \tag{3.9}$$

Definition 5 (Upper & Lower Bound) *Let Ω be a universe, $X \in \mathcal{P}(\Omega)$ be a set in the universe and $A \subset \mathcal{P}(U), A \neq \emptyset$, non-empty subsets in the universe.*

$$X \textit{ is an } \text{upper bound} \textit{ for } A :\Leftrightarrow \forall Y \in A : Y \subset X \tag{3.10}$$

$$X \textit{ is a } \text{lower bound} \textit{ for } A :\Leftrightarrow \forall Y \in A : X \subset Y \tag{3.11}$$

We also define:

$$\mathbf{U}_A := \{U \in \mathcal{P}(\Omega) | \forall Y \in A : Y \subset U\} \tag{3.12}$$

$$\mathbf{L}_A := \{L \in \mathcal{P}(\Omega) | \forall Y \in A : L \subset Y\} \tag{3.13}$$

as sets of all upper/lower bounds for A.

Because our definition of upper and lower bounds is based on power-sets, existence is guaranteed: Given an arbitrary set S, the S and $\emptyset$ are always elements of $\mathcal{P}(S)$. For each element Y of a non-empty selection $A \subset \mathcal{P}(S)$ of the power-set, $Y \subset S$ and $\emptyset \subset Y$ holds. So S is an upper and $\emptyset$ is a lower bound for A.

Definition 6 (Supremum & Infimum) *Let Ω be a universe, $X \in \mathcal{P}(\Omega)$ be sets in the universe and $A \subset \mathcal{P}(\Omega), A \neq \emptyset$ a non-empty selection of sets in the universe. If*

$$X = \sup A := \bigcup_{Y \in A} Y :\Leftrightarrow X \in \mathbf{U}_A \wedge \forall U \in \mathbf{U}_A : X \subset U \tag{3.14}$$

$$X = \inf A := \bigcap_{Y \in A} Y :\Leftrightarrow X \in \mathbf{L}_A \wedge \forall L \in \mathbf{L}_A : L \subset X \tag{3.15}$$

then X is called supremum/infimum *for A.*

Existence for supremum and infimum is guaranteed, because upper and lower bounds exist as shown above. Thus, for any non-empty selection $A \subset \mathcal{P}(S)$ of a power-set, $\mathbf{U}_A$ and $\mathbf{L}_A$ are not empty. So we need to proof, that $X = \bigcup_{Y \in A} Y$ respectively $X = \bigcap_{Y \in A} Y$ are in fact the smallest upper and the largest lower bound. Or in other words: Does another bound $X' \in \mathbf{U}_A$ respectively $X' \in \mathbf{L}_A$ exist with $X' \neq X$ and $X' \subset X$ respectively $X \subset X'$?

1. Supremum: We assume $X' \in \mathbf{U}_A$ with $X' \neq X$ and $X' \subset X$ for $X = \bigcup_{Y \in A} Y$ exists, then an element $x \in X$ exists, which is not element of X'. Because X

is the union of all sets in selection A, x must be element of at least one of its sets. However, then X' cannot include sets containing x. Thus, X' cannot be an upper bound for A and $X = \sup A$.

2. Infimum: We assume $X' \in \mathbf{L}_A$ with $X' \neq X$ and $X \subset X'$ for $X = \bigcap_{Y \in A} Y$ exists, then an element $x \in X'$ exists, which is not element of X. Because X is the intersection of all sets in selection A, x cannot be element of at least one of its sets. However, then X' must include sets containing x. Thus, X' cannot be a lower bound fo A and $X = \inf A$.

Supremum and Infimum are unique for any non-empty selection of sets in a universe and can be obtained by its union respectively its intersection. [13]

Definition 7 (Cartesian Product) *Let U be a universe and $X_n \in \mathcal{P}(U)$ sets with $i = 1...n, n \in \mathbb{N}$, then:*

$$X_1 \times ... \times X_n := \{(x_1, ..., x_n)\} \tag{3.16}$$

is called Cartesian product.

Definition 8 (Relation) *A relation is a subset of a Cartesian product:*

$$R \subset X_1 \times ... \times X_n \tag{3.17}$$

The relation of only two sets is called binary relation. *Instead of writing $(x, y) \in R$ we may also use the shorter notation xRy.*

An arbitrary relation $R \subset A \times B$ is called *homogeneous* if $A = B$, otherwise it is called *heterogeneous*. However, an arbitrary relation $R \subset A \times B$ is also homogeneous in the sense of $R \subset A \times B \subset (A \cup B) \times (A \cup B)$. For the remainder of this section we focus on homogeneous relations unless noted otherwise.

In order to clarify our notation, when we are specifically working with relations instead of ordinary sets, we use the symbols $\sqsubset$ for inclusion,$\not\sqsubset$ for non-inclusion, $\sqcup$ for union and $\sqcap$ for intersection, i.e.:

$$R \sqsubset S :\Leftrightarrow \forall x, y[(x, y) \in R \rightarrow (x, y) \in S] \tag{3.18}$$

$$R \not\sqsubset S :\Leftrightarrow \exists x, y[(x, y) \in R \wedge (x, y) \notin S] \tag{3.19}$$

$$R \sqcup S := \{(x, y)|(x, y) \in R \vee (x, y) \in S\} \tag{3.20}$$

$$R \sqcap S := \{(x, y)|(x, y) \in R \wedge (x, y) \in S\} \tag{3.21}$$

Furthermore, we use $\mathcal{R}(A)$ to denote the set of all homogeneous relations in set A and the symbols $\mathcal{O}$ and $\mathcal{A}$ to denote the empty relation and the universal relation:

$$\mathcal{R}(A) := \{R|R \subset A \times A\} \tag{3.22}$$

$$(\mathcal{O} := \emptyset) \subset A \times A \qquad \Leftrightarrow \forall R \in \mathcal{R}(A)[\mathcal{O} \sqsubset R] \tag{3.23}$$

$$(\mathcal{A} := A \times A) \subset A \times A \qquad \Leftrightarrow \forall R \in \mathcal{R}(A)[R \sqsubset \mathcal{A}] \tag{3.24}$$

Definition 9 (Composition of Binary Relations) *Let $R, S \in \mathcal{R}(A)$. Then $R \circ S \in \mathcal{R}(A)$ is defined*

$$R \circ S = RS := \{(r, s) \in A \times A|\exists x \in A : (r, x) \in R \wedge (x, s) \in S\} \tag{3.25}$$

and called composition *or* multiplication *of R and S. Instead of writing $R \circ S$ we also write simply RS.*

In conjunction with $\sqsubset$ composition is monotone:

$$\forall P, Q, R \in \mathcal{R}(A) : P \sqsubset Q \rightarrow R \circ P \sqsubset R \circ Q \tag{3.26}$$

$$\forall P, Q, R \in \mathcal{R}(A) : P \sqsubset Q \rightarrow P \circ R \sqsubset Q \circ R \tag{3.27}$$

- ($\Rightarrow$) Following the definition of composition, it can easily be observed, if Q includes P, all elements of P are also in Q:

$$\forall x, y : (x, y) \in R \circ P \tag{3.28}$$

$$\rightarrow \exists z : (x, z) \in R \wedge (z, y) \in P \overset{P \sqsubset Q}{\rightarrow} \exists z : (x, z) \in R \wedge (z, y) \in Q \tag{3.29}$$

$$\rightarrow (x, y) \in R \circ Q \tag{3.30}$$

An analogous deduction can be shown for the right hand side composition of R.

- ($\Leftarrow$) The opposite direction can be proven indirectly, assuming $R \circ P \sqsubset R \circ Q$ holds, but $P \sqsubset Q$ does not:

Definition 10 (Identity Relation) *The relation $\mathcal{I} \in \mathcal{R}(A)$*

$$\mathcal{I} := \{(a,b) \in A \times A | a = b\} = \{(a,a) | a \in A\} \subset A \times A \tag{3.31}$$

is called identity relation*.*

$(\mathcal{R}(A), \circ, \mathcal{I})$ is a monoid, i.e. for all relations in $\mathcal{R}(A)$ composition is associative and $\mathcal{I}$ serves as it's identity element:

$$(Q \circ R) \circ S = Q \circ (R \circ S) \tag{3.32}$$

$$R \circ \mathcal{I} = \mathcal{I} \circ R = R \tag{3.33}$$

Also, $\mathcal{O}$ serves as absorbing element for composition:

$$R \circ \mathcal{O} = \mathcal{O} \circ R = \mathcal{O} \tag{3.34}$$

Because $(\mathcal{R}(A), \circ, \mathcal{I})$ is a monoid, we can define exponentiation:

Definition 11 (Exponentiation of Relations) *Let $R \in \mathcal{R}(A)$ and $n \in \mathbb{N}$.*

$$R^0 := \mathcal{I} \tag{3.35}$$

$$R^n := R \circ R^{n-1} \tag{3.36}$$

Consider the following example:

$$A = \{a, b, c, d\} \tag{3.37}$$

$$R = \{(a,b), (a,c), (c,d)\} \tag{3.38}$$

$$R^0 = \{(a,a), (b,b), (c,c), (d,d)\} \tag{3.39}$$

$$R^1 = R \circ R^0 = R \circ \mathcal{I} = \{(a,b), (a,c), (c,d)\} \tag{3.40}$$

$$R^2 = R \circ R^1 = R \circ R = \{(a,d)\} \tag{3.41}$$

$$R^3 = R \circ R^2 = \mathcal{O} \tag{3.42}$$

$$R^4 = R^5 = R^6 = ... = R \circ \mathcal{O} = \mathcal{O} \tag{3.43}$$

Definition 12 (Reflexivity) *A relation $R \in \mathcal{R}(A)$ is called:*

$$\text{reflexive} :\Leftrightarrow \forall a \in A : (a,a) \in R \tag{3.44}$$

$$\text{irreflexive} :\Leftrightarrow \forall a \in A : (a,a) \notin R \tag{3.45}$$

Definition 13 (Reflexive Closure) *Let $R \in \mathcal{R}(A)$. Then*

$$R^{\circ} := \inf\{S|R \sqsubset S \wedge S \textit{ reflexive }\} = R \sqcup \mathcal{I} \tag{3.46}$$

is called reflexive closure *of R.*

The reflexive closure of a homogeneous relation R is the infimum or largest lower bound of the set $A = \{S|R \sqsubset S \wedge S \text{ reflexive }\}$ containing all reflexive relations, which include R. The smallest reflexive relation is $\mathcal{I}$. The smallest relation including R is R itself. So, for an arbitrary relation $R' \in A$ the inclusions $R \sqsubset R'$, $\mathcal{I} \sqsubset R'$ and $(R \sqcup \mathcal{I}) \sqsubset R'$ hold. Thus $R \sqcup \mathcal{I}$ is a lower bound for A and $(R \sqcup \mathcal{I}) \sqsubset \inf A$ holds. Vice versa, $R \sqcup \mathcal{I}$ is an element of A and any relation $R'' \sqsubset (R \sqcup \mathcal{I})$ does either not include R or is not reflexive. Thus $R \sqcup \mathcal{I}$ is also the smallest relation in A and $\inf A \sqsubset (R \sqcup \mathcal{I})$. From $(R \sqcup \mathcal{I}) \sqsubset \inf A$ and $\inf A \sqsubset (R \sqcup \mathcal{I})$ follows $\inf A = (R \sqcup \mathcal{I})$. [13]

Definition 14 (Symmetry) *A relation $R \in \mathcal{R}(A)$ is called:*

$$\text{symmetric} :\Leftrightarrow \forall a,b \in A : (a,b) \in R \Rightarrow (b,a) \in R \tag{3.47}$$

$$\text{asymmetric} :\Leftrightarrow \forall a,b \in A : (a,b) \in R \Rightarrow (b,a) \notin R \tag{3.48}$$

$$\text{antisymmetric} :\Leftrightarrow \forall a,b \in A : (a,b) \in R \wedge (b,a) \in R \Rightarrow a = b \tag{3.49}$$

Definition 15 (Transitivity) *A relation $R \in \mathcal{R}(A)$ is called:*

$$\text{transitive} :\Leftrightarrow R^2 \sqsubset R \tag{3.50}$$

$$:\Leftrightarrow \forall a,b,c \in A : (a,b) \in R \wedge (b,c) \in R \rightarrow (a,c) \in R \tag{3.51}$$

$$\text{intransitive} :\Leftrightarrow R^2 \not\sqsubset R \tag{3.52}$$

$$:\Leftrightarrow \forall a,b,c \in A : (a,b) \in R \wedge (b,c) \in R \rightarrow (a,c) \notin R \tag{3.53}$$

The equivalent characterization of transitivity is $R^2 \sqsubset R$ is easily obtained:

$$\forall a, b, c \in A : (a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R \tag{3.54}$$

$$\Leftrightarrow \forall a, c \in A : (a, c) \in RR \rightarrow (a, c) \in R \tag{3.55}$$

$$\Leftrightarrow \forall a, c \in A : (a, c) \in R^2 \rightarrow (a, c) \in R \tag{3.56}$$

$$\Leftrightarrow R^2 \sqsubset R \tag{3.57}$$

If $\mathcal{T}(A) := \{R \in \mathcal{R}(A) | R^2 \sqsubset R\}$ is the set over all transitive relations of set A, it's infimum $I := \inf \mathcal{T}(A)$ is also transitive. Assuming it is not, at least one element exists, which is in I^2, but not in I. Because I is the infimum of $\mathcal{T}(A)$, all transitive relations R must include it. Thus the same element is in R^2, but not in R. However, this is a contradiction, because R is transitive and all elements in R^2 must be in R.

$$\neg(I^2 \sqsubset I) \Leftrightarrow \forall x, y : \neg[(x, y) \in I^2 \rightarrow (x, y) \in I] \tag{3.58}$$

$$\Leftrightarrow \forall x, y : (x, y) \in I^2 \wedge (x, y) \notin I \tag{3.59}$$

$$\Leftrightarrow \forall x, y \exists z : (x, z) \in I \wedge (z, y) \in I \wedge (x, y) \notin I \tag{3.60}$$

$$\Leftrightarrow \forall x, y \exists z : (x, z) \in R \wedge (z, y) \in R \wedge (x, y) \notin R \tag{3.61}$$

$$\Leftrightarrow \forall x, y : (x, y) \in R^2 \wedge (x, y) \notin R \tag{3.62}$$

$$\Leftrightarrow \forall x, y : \neg[(x, y) \in R^2 \rightarrow (x, y) \in R] \tag{3.63}$$

$$\Leftrightarrow \neg(R^2 \sqsubset R) \lightning \tag{3.64}$$

Definition 16 (Transitive Closure) *Let $R \in \mathcal{R}(A)$ and $i \in \mathbb{N}$. Then*

$$R^+ := \inf\{S | R \sqsubset S \wedge S \textit{ transitive }\} = \sup\{R^i | i \geq 1\} \tag{3.65}$$

is called transitive closure *of R.*

The transitive closure of a relation is the infimum or greatest lower bound of the set $A = \{S | R \sqsubset S \wedge S \text{ transitive }\}$ containing all transitive relations, which include R. Because A contains only transitive relations, its infimum $I = \inf A$ is also transitive.

Definition 17 (Transitive-Reflexive Closure) *$R^* := R^+ \cup I$ is called reflexive-transitive closure*

Definition 18 (Order Relation) *content...*

3.3 Mereology

Mereology is the logical study on the semantics of parthood. *"As a formal theory, mereology is simply an attempt to set out the general principles underlying the relationships between a whole and its constituent parts [...]"* [16]. Achille C. Varzi describes a collection of formal theories, i.e. sets of distinct axioms, of mereology in [16], which will be summarized in this section.

ToDo: add SEP reference [15]

ToDo: add paragraph with applications of mereology in computer science [12]

The first part of this section will just introduce the axioms of mereology. Then, at the end of this section, we will use these axiom to build some of the theories described in [16]. Note, that the terms relation, relationship and predicate may be used synonymously throughout this section.

3.3.1 Parthood

First we define the intuitive notion of the parthood relationship:

Definition 19 (partOf) *Let x and y objects of interest. We define:*

$$x \textit{ partOf } y :\Leftrightarrow x \textit{ is a constituent part of } y \tag{3.66}$$

We further assume, that partOf satisfies the following properties:

$$\text{(P1)} \quad x \text{ partOf } x \qquad \text{(Reflexivity)} \tag{3.67}$$

$$\text{(P2)} \quad x \text{ partOf } y \wedge y \text{ partOf } x \rightarrow x = y \qquad \text{(Antisymmetry)} \tag{3.68}$$

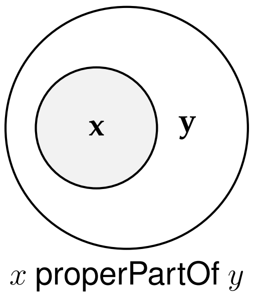
$$\text{(P3)} \quad x \text{ partOf } y \wedge y \text{ partOf } z \rightarrow x \text{ partOf } z \qquad \text{(Transitivity)} \tag{3.69}$$

Thus, partOf induces a partial order of things.

However, since the reflexive parthood may be to week for some cases, we also define a stricter, irreflexive parthood relationship as follows:

$$x \text{ properPartOf } y :\Leftrightarrow x \text{ partOf } y \wedge \neg(y \text{ partOf } x) \qquad \text{(Proper Part)} \tag{3.70}$$

Proper parthood induces a strict partial order of things. Figure 3.1 shows a schematic illustration of proper parts.



This Venn-style diagram depicts a schematic illustration of Proper Part: x is certainly a part of y, however y is not a part of x.

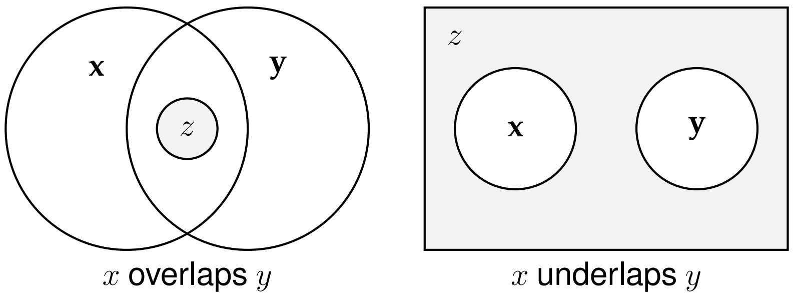
Figure 3.1 A schematic depiction of Proper Part

In addition to the relationships above we introduce the following predicates in order to provide a more concise notation:

$$x \text{ overlaps } y :\Leftrightarrow \exists z : z \text{ partOf } x \land z \text{ partOf } y \qquad \text{(Overlap)} \qquad (3.71)$$

$$x \text{ underlaps } y :\Leftrightarrow \exists z : x \text{ partOf } z \land y \text{ partOf } z \qquad \text{(Underlap)} \qquad (3.72)$$

Overlap models situations, where two things share at least on distinct part. Underlap models situations, where two things are part of the same distinct thing. Figure 3.2 illustrates both in a schematic fashion.



These Venn-style diagrams depict schematic illustrations of Overlap & Underlap:

- x overlaps y: x and y share a distinct part z, which is emphasized as gray area.
- x underlaps y: x and y are both parts of z, which is emphasized as gray area.

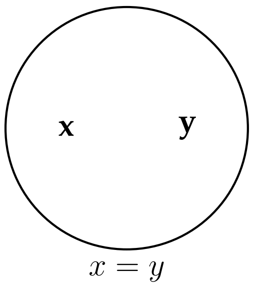
Figure 3.2 A schematic depiction of Overlap & Underlap

ToDo: Define and outline Identity/Equality. Note seems to be some problems with mereological equality of words, see [15]

In axiom 3.68 (P2) we already used identity/equality "=" to describe antisymmetry. So we used first order logic *with* identity/equality to define parthood. However, it is worthwhile noting, that parthood itself also allows for defining identity/equality [15]:

$$x = y :\Leftrightarrow x \text{ partOf } y \wedge y \text{ partOf } x \qquad \text{(Identity/Equality)} \tag{3.73}$$

Note that identity and equality cannot be distinguished in the mereological context. Figure 3.3 shows a schematic depiction of identity/equality.



This Venn-style diagram depicts a schematic illustration of Identity/Equality.

Figure 3.3 A schematic depiction of Identity/Equality

In any case, the right hand side of 3.73 defines an equivalence relation due to partOf being a partial order.

Proof 1 *Let $\prec$ be a partial order (reflexive, antisymmetric, transitive) and*

$$x \approx y :\Leftrightarrow x \prec y \wedge y \prec x$$

then $\approx$ is an equivalence relation (reflexive, symmetric, transitive). If $\approx$ is not an equivalence relation, then it is not reflexive, not symmetric or not transitive. However:

1. *if $\approx$ is not reflexive:*

$$\neg(x \approx x) \Leftrightarrow \neg(x \prec x \wedge x \prec x) \Leftrightarrow \neg(x \prec x) \qquad ↯$$

2. *if $\approx$ is not symmetric:*

$$\begin{aligned}
&\neg(x \approx y \to y \approx x)\\
&\Leftrightarrow x \approx y \land \neg(y \approx x)\\
&\Leftrightarrow (x \prec y \land y \prec x) \land \neg(y \prec x \land x \prec y)\\
&\Leftrightarrow (x \prec y \land y \prec x) \land \neg(x \prec y \land y \prec x)\\
&\Leftrightarrow (x \prec x) \land \neg(x \prec x) \qquad \lightning
\end{aligned}$$

3. *if $\approx$ is not transitive:*

$$\begin{aligned}
&\neg(x \approx z \land z \approx y \to x \approx y)\\
&\Leftrightarrow x \approx z \land z \approx y \land \neg(x \approx y)\\
&\Leftrightarrow (x \prec z \land z \prec x) \land (z \prec y \land y \prec z) \land \neg(x \prec y \land y \prec x)\\
&\Leftrightarrow (x \prec z \land z \prec y \land y \prec z \land z \prec x) \land \neg(x \prec y \land y \prec x)\\
&\Leftrightarrow (x \prec y \land y \prec x) \land \neg(x \prec y \land y \prec x)\\
&\Leftrightarrow (x \prec x) \land \neg(x \prec x) \qquad \lightning
\end{aligned}$$

We shall later see that identity/equality can also be extensionally determined:

$$x = y \Leftrightarrow \forall z(z \textsf{ partOf } x \leftrightarrow z \textsf{ partOf } y) \tag{3.74}$$

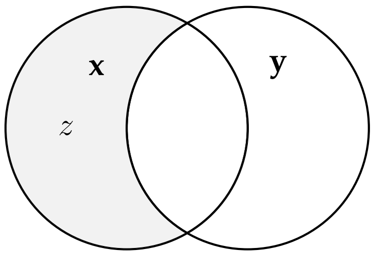
i.e. two things x and y identical or equal if and only if all parts of x are parts of y and vice versa.

3.3.2 Supplementation

The fourth axiom, which can be assumed in an universe described by means of mereology, is the supplementation axiom. It models the effects of situations more precisely, where one thing is not part of another. Namely, if this is the case, a third thing may exist, which is part of the former but not part of the latter.

$$\text{(P4)} \qquad \neg(x \textsf{ partOf } y) \to \exists z(z \textsf{ partOf } x \land \neg(z \textsf{ overlaps } y)) \tag{3.75}$$

The supplementation axiom reads: If a thing x is not part of another thing y, then at least one part of x does not share further parts with y. Figure 3.4 depicts a schematic example of the supplementation axiom.



This Venn-style diagram exemplifies the supplementation axiom: x is not part of y. z is emphasized as gray area. x contains z, but z shares no further parts with y.

Figure 3.4 A schematic example of the Supplementation Axiom

ToDo: add paragraph explaining "strong" and "weak" supplementation

One can also observe from figure 3.4, that the supplementation axiom can be interpreted as analogue to set-theoretic difference.

3.3.3 Sum, Product & Difference

The next three axioms allow for the notions of sum, product and difference in a mereological context.

3.3.3.1 Sum

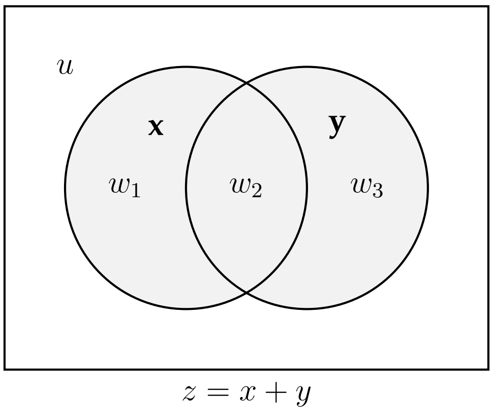
The fifth axiom, which can be assumed in an universe described by means of mereology, is the sum axiom. It models the intuitive notion, where all parts of one thing and all parts of another thing are exactly the constituent parts of a third thing.

$$\text{(P5)} \quad \begin{aligned} &x \text{ underlaps } y \\ &\rightarrow \exists z \forall w (w \text{ overlaps } z \leftrightarrow (w \text{ overlaps } x \vee w \text{ overlaps } y)) \end{aligned} \tag{3.76}$$

The sum axiom reads: If things x and y are two parts of the same thing, then another thing z exist, which only shares parts with things, which in turn share parts with x or y. Then z can be interpreted as the sum of x and y. This notion is captured by following definition of the term $z = x + y$:

$$z = x + y :\Leftrightarrow \forall w(w \text{ overlaps } z \leftrightarrow (w \text{ overlaps } x \vee w \text{ overlaps } y)) \tag{3.77}$$

Figure 3.5 depicts a schematic example of the sum axiom.



This Venn-style diagram exemplifies the sum axiom: x and y are part of u. $z = x + y$ is emphasized as gray area. All w_i share parts with z, if and only if they share parts with x or y.

Figure 3.5 A schematic example of the Sum Axiom

One can also observe similarities between the sum axiom and set-theoretic union from figure 3.5 and the definition 3.77.

3.3.3.2 Product

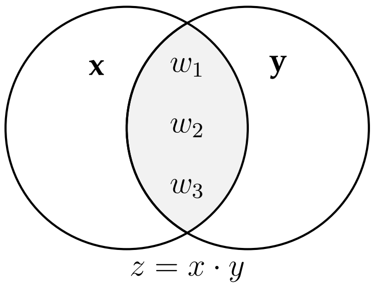
The sixth axiom, which can be assumed in an universe described by means of mereology, is the product axiom. It models the intuitive notion, where all things, which are parts of two things at the same time, are exactly the constituent parts of a third thing.

$$\text{(P6)} \quad \begin{aligned} &x \text{ overlaps } y \\ &\rightarrow \exists z \forall w(w \text{ partOf } z \leftrightarrow (w \text{ partOf } x \wedge w \text{ partOf } y)) \end{aligned} \tag{3.78}$$

The sum axiom reads: If things x and y share at least one part, then another thing z exists, which only consists of parts, which in turn are parts of x and y at the same time. Then z can be interpreted as the product of x and y. This notion is captured by the following definition of the term $z = x \cdot y$:

$$z = x \cdot y :\Leftrightarrow \forall w(w \textsf{ partOf } z \leftrightarrow (w \textsf{ partOf } x \wedge w \textsf{ partOf } y)) \tag{3.79}$$

Figure 3.6 depicts a schematic example of the product axiom.



This Venn-style diagram exemplifies the product axiom: x and y share arts. $z = x \cdot y$ is emphasized as gray area. All w_i are part of z, if and only if they are part of x and y.

Figure 3.6 A schematic example of the Product Axiom

One can also observe similarities between the sum axiom and set-theoretic intersection from figure 3.6 and the definition 3.79.

3.3.3.3 Difference

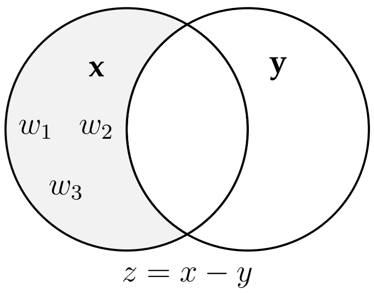
The seventh axiom, which can be assumed in an universe described by means of mereology, is the difference axiom. It models the intuitive notion, where things may be split, so that their constituent parts are regrouped into disjoint things, which share no parts.

$$\text{(P7)} \quad \begin{aligned} &\exists z(z \textsf{ partOf } x \wedge \neg(z \textsf{ overlaps } y)) \\ &\to \exists z \forall w(w \textsf{ partOf } z \leftrightarrow (w \textsf{ partOf } x \wedge \neg(w \textsf{ overlaps } y))) \end{aligned} \tag{3.80}$$

The difference axiom reads: If for things x and y a supplementary thing exists, which is part of x, but shares no parts with y, then another thing z exists, consisting of parts, which are also part of x, but share no parts with y. Then z can be interpreted as the difference between x and y, in that order. This notion is captured by the following definition of the term $z = x - y$:

$$z = x - y :\Leftrightarrow \forall w(w \text{ partOf } z \leftrightarrow (w \text{ partOf } x \wedge \neg(w \text{ overlaps } y))) \tag{3.81}$$

Figure 3.7 depicts a schematic example of the difference axiom.



This Venn-style diagram exemplifies the difference axiom: x is not part of y and parts of x exists, which do not share parts with y. $z = x - y$ is emphasized as gray area. All w_i are part of z, if and only if they are part of x and do not share further parts with y.

Figure 3.7 A schematic example of the Difference Axiom

One can also observe again similarities between the sum axiom and set-theoretic difference from figure 3.7 and the definition 3.81.
ToDo: Outline dependencies with supplementation axiom

3.3.4 Universal Top, Complement & Bottom

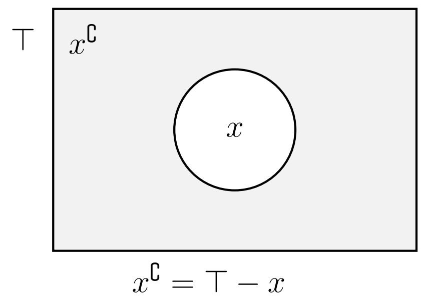
The sum axiom (P5) at 3.76 gives immediate rise to the idea, that all things can be summed up to an universal thing and all things are part of it. This notion is captured by the top axiom:

$$\exists \top \forall x(x \text{ partOf } \top) \qquad \text{(Top)} \tag{3.82}$$

Having an universe also facilitates the definition of a universal or absolute complement:

$$x^{\complement} := \top - x \qquad \text{(Complement)} \tag{3.83}$$

Figure 3.8 depicts a schematic illustration of an universal top and a complement.



This Venn-style diagram illustrates universal top and a complement: Top $\top$ is the rectangle containing everything. The complement x^{C} of x is emphasized as gray area.

Figure 3.8 A schematic illustration of Universal Top & Complement

An universal top renders the underlap relationship as defined at 3.72 trivially true, thus the mereological sum of two things as defined at 3.77 can never be undefined. In an algebraic sense, an universal top also provides an absorbing element for mereolgical sum and a neutral element for the mereoligcal product as defined at 3.79:

$$\top = \top + x = x + \top \tag{3.84}$$

$$x = \top \cdot x = x \cdot \top \tag{3.85}$$

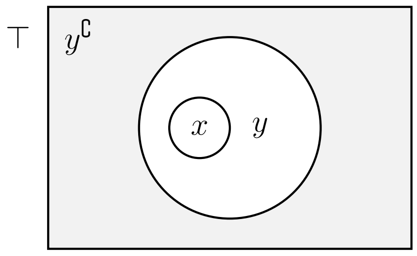
Properties of the mereological complement can in part be observed from figure 3.8. Obviously, in addition to the definition of complements, the following identity also holds:

$$x = \top - x^{\mathsf{C}} \tag{3.86}$$

Less obvious properties may be the following equivalences incorporating complement, parthood and overlap:

$$\begin{gathered}\forall x \forall y (x \text{ partOf } y^{\mathsf{C}} \leftrightarrow \neg(x \text{ overlaps } y)) \\ \forall x \forall y (x \text{ partOf } y \leftrightarrow \neg(x \text{ overlaps } y^{\mathsf{C}}))\end{gathered} \tag{3.87}$$

Figure 3.9 shows a schematic illustration of this equivalence. Both equations fol-



$$x \ \mathsf{partOf}\ y^{\complement} \leftrightarrow \neg(x \ \mathsf{overlaps}\ y)$$

This Venn-style diagram illustrates the equivalence incorporating complement, parthood and overlap: Top $\top$ is the rectangle containing everything. x is a proper part of y. The complement $y^{\complement}$ of y is emphasized as gray area. x shares no parts with $y^{\complement}$.

Figure 3.9 A schematic illustration the equivalence incorporating Complement, Parthood & Overlap

low directly from the definition of mereological difference by simple deduction, since x partOf $\top$ is always true:

$$\begin{aligned} x^{\complement} = \top - x &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x^{\complement} \leftrightarrow x \ \mathsf{partOf}\ \top \wedge \neg(y \ \mathsf{overlaps}\ x)) \\ &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x^{\complement} \leftrightarrow \mathit{true} \wedge \neg(y \ \mathsf{overlaps}\ x)) \\ &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x^{\complement} \leftrightarrow \neg(y \ \mathsf{overlaps}\ x)) \end{aligned} \tag{3.88}$$

$$\begin{aligned} x = \top - x^{\complement} &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x \leftrightarrow x \ \mathsf{partOf}\ \top \wedge \neg(y \ \mathsf{overlaps}\ x^{\complement})) \\ &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x \leftrightarrow \mathit{true} \wedge \neg(y \ \mathsf{overlaps}\ x^{\complement})) \\ &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ x \leftrightarrow \neg(y \ \mathsf{overlaps}\ x^{\complement})) \end{aligned} \tag{3.89}$$

A direct corollary is the identity of complement involution:

$$(x^{\complement})^{\complement} = x \tag{3.90}$$

This identity is deduced by substitution of the equivalence above at 3.88 and 3.89:

$$\begin{aligned} (x^{\complement})^{\complement} = T - x^{\complement} &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ (x^{\complement})^{\complement} \leftrightarrow \neg(y \ \mathsf{overlaps}\ x^{\complement})) \\ &\Leftrightarrow \forall y(y \ \mathsf{partOf}\ (x^{\complement})^{\complement} \leftrightarrow y \ \mathsf{partOf}\ x) \\ &\Leftrightarrow (x^{\complement})^{\complement} = x \end{aligned} \tag{3.91}$$

This property allows for a refinement of the definition for mereological difference, which again shows another similarity with set-theoretic difference:

$$z = x - y = x \cdot y^{\complement} \tag{3.92}$$

The proof is the same simple deduction as above:

$$\begin{aligned} z = x - y &\Leftrightarrow \forall w(w \text{ partOf } z \leftrightarrow w \text{ partOf } x \wedge \neg(w \text{ overlaps } y)) \\ &\Leftrightarrow \forall w(w \text{ partOf } z \leftrightarrow w \text{ partOf } x \wedge w \text{ partOf } y^{\complement}) \\ &\Leftrightarrow z = x \cdot y^{\complement} \end{aligned} \tag{3.93}$$

The notion of an universal top gives immediate rise to the question, whether a converse thing, an universal bottom, exists. The universal bottom for parthood satisfies:

$$\exists \bot \forall x(\bot \text{ partOf } x) \qquad \text{(Bottom)} \tag{3.94}$$

An universal bottom renders the overlap relationship as defined at 3.71 trivially true, thus the mereological product of two things can never be undefined. In an algebraic sense, an universal bottom provides an absorbing element for the mereological product and a neutral element for the mereological sum:

$$\bot = \bot \cdot x = x \cdot \bot \tag{3.95}$$

$$x = \bot + x = x + \bot \tag{3.96}$$

Moreover, it makes mereological difference sound. With an universal bottom the difference $x - x$ of only one thing is now possible:

$$\begin{aligned} z = x - x &\Leftrightarrow z = x \cdot x^{\complement} \\ &\Leftrightarrow \forall y(y \text{ partOf } z \leftrightarrow y \text{ partOf } x \wedge y \text{ partOf } x^{\complement}) \\ &\Leftrightarrow z = \bot \end{aligned} \tag{3.97}$$

The only thing, which is part of x and its complement at the same time, is the universal bottom. Given this fact, now complements of top and bottom can also be determined:

$$\top^{\mathtt{C}} = \top - \top = \bot \tag{3.98}$$

$$\bot^{\mathtt{C}} = \top - \bot = \top - \top^{\mathtt{C}} = \top \cdot (\top^{\mathtt{C}})^{\mathtt{C}} = \top \cdot \top = \top \tag{3.99}$$

However, the notion of an universal bottom or null thing is more controversial than the notion of an universal top [16].
ToDo: Outline why bottom is disputed. Perhaps, if bottom and atoms are assumed, bottom would be the only "real" atom.

3.3.5 Unrestricted Fusion

3.3.6 Atomic Parts

3.3.7 Mereology Theories

3.4 Formal Languages & Grammars

[14]

3.4.1 Context-Free Languages & Grammars

3.5 Traceability

[17] [5] [8] TBD.

3.5.1 Traceability Relationship

3.5.2 Traceability Link

3.5.3 Traceability Recovery

3.5.4 Traceability Exploration

3.6 Ontologies

TBD. [6]

3.7 Megamodeling

TBD. [1] [3]

3.7.1 MegaL

[10] [2] [9]

3.7.1.1 MegaL/Xtext

[7]

3.8 Program Analysis

TBD.

3.9 XML Data Binding

TBD.

3.9.1 Java Architecture for XML Binding (JAXB)

3.10 Object Relational Mapping

TBD.

3.10.1 Java Persistence API (JPA)

3.10.2 Hibernate

3.11 Another Tool For Language Recognition (ANTLR)

[11]

Chapter 4

Hypotheses

This chapter introduces the hypotheses this thesis is bases upon. TBD.

4.1 Fragments

The Oxford English Dictionary defines fragment[1] as *"[...] isolated or incomplete part of something [...]"*. Regarding natural languages, sentence fragments are incomplete sentences lacking one or more grammatical elements. **ToDo: Lacks citation for "sentence fragment"!**

The intuitive notion of a fragment when working with and/or talking over computer programs is, that it is a piece of code containing a functionality of momentary interest. In that regard, we exclusively use the word fragment referring to isolated or incomplete parts of things having a textual representation, e.g. computer programs. However, computer science knows different levels of organization in text, which effects the semantics of textual fragments.

The most basic level of textual organization is no organization at all. Random text is only restricted by its alphabet, which allows us to specify fragmentation solely in terms of concatenation.

[1] `https://en.oxforddictionaries.com/definition/fragment` (retrieved 08/28/2017)

Definition 20 (Fragments over Alphabets) *Let Σ be an alphabet and $x, y \in \Sigma^*$ be two words over that alphabet. x is a fragment of y, if and only if there is a concatenation containing x and resulting in y:*

$$\textit{fragmentOf} : \Sigma^* \times \Sigma^* \tag{4.1}$$

$$x \textit{ fragmentOf } y :\Leftrightarrow \exists u, v \in \Sigma^*(uxv = y) \tag{4.2}$$

We can also observe from definition 20 that textual fragmentation shares similarities with mereological parthood, i.e. the fragmentOf definition is also reflexive, antisymmetric and transitive:

1. Reflexivity: x fragmentOf x
   x fragmentOf x holds for all x since $u = v = \epsilon \in \Sigma^*$.

2. Antisymmetry: x fragmentOf $y \wedge y$ fragmentOf $x \rightarrow x = y$
   x fragmentOf $y \wedge y$ fragmentOf x is equivalent to $axb = y \wedge cyd = x$, from which follows $caxbd = x$ and then $a = b = c = d = \epsilon$, thus $x = y$ follows eventually

3. Transitivity: x fragmentOf $y \wedge y$ fragmentOf $z \rightarrow x$ fragmentOf z
   x fragmentOf $y \wedge y$ fragmentOf z is equivalent to $axb = y \wedge cyd = z$, from which follows $caxbd = z$, which is in turn equivalent to $uxv = z$ with $u = ca$ and $v = bd$, thus x fragmentOf z follows eventually

There is a universal bottom element, which is fragment of all words:

$$\forall x \in \Sigma^*(\epsilon \text{ fragmentOf } x) \tag{4.3}$$

and there is a universal top element, all words are fragments of:

$$\begin{aligned} &\forall x \in \Sigma^*(x \text{ fragmentOf } W) \\ &W := w_1 w_2 w_3 ... \qquad w_i \in \Sigma^* \backslash \{\epsilon\}, i \in \mathbb{N} \end{aligned} \tag{4.4}$$

where W is the concatenation of all words in Σ^* and thus is an infinite word. $W \in \Sigma^\omega$ is interpreted as mapping $\mathbb{N} \rightarrow \Sigma$, giving a symbol from the alphabet for its position in the word. **ToDo: citation for infinite words, Handbook on Formal languages**

We can now easily agree upon, that fragmentOf is partOf for texts. Thus both are logically equivalent relations:

$$\text{fragmentOf} \equiv \text{partOf} \tag{4.5}$$

ToDo: refer to philosophical problems from SEP

A higher level of textual organization are formal languages, i.e. subsets of Σ^* for any alphabet Σ [14]. Given any language specified with set-builder notation:

$$L := \{w \in \Sigma^* | \Phi(w)\} \subseteq \Sigma^* \tag{4.6}$$

where $\Phi(w)$ is an arbitrary restriction on w, we can still observe the same behavior regarding fragments, e.g.:

$$L = \{a^n b^n | n \in \mathbb{N} : n \geq 2\} \subseteq \{a, b\}^* = \{aabb, aaabbb, aaaabbbb, ...\}$$

$$...$$

$$ab \text{ fragmentOf } aabb$$

$$aabb \text{ fragmentOf } aaabbb$$

$$aaabbb \text{ fragmentOf } aaaabbbb$$

$$...$$

However, we can only observe the behavior partly for fragmentOf $: \Sigma^* \times \Sigma^*$ and fragmentOf $: \Sigma^* \times L$, but not for fragmentOf $: L \times L$. In other words, the fragments of words in formal languages are not necessarily words of that language itself. This becomes even more apparent if we consider fragments and languages generated by formal grammars.

Definition 21 (Fragments generated by Grammars) *Let $G = (V, \Sigma, P, S)$ be a grammar and $x, y \in \Sigma^*$ be two words over its terminal symbols. x is a fragment of y generated by G, if and only if there is a derivation from a concatenation containing x to y:*

$$x \text{ fragmentOf}_G\ y \Leftrightarrow \exists u, v \in (V \cup \Sigma)^* (S \Rightarrow_G^* uxv \Rightarrow_G^* y) \tag{4.7}$$

4.1.1 Fragment Languages

4.2 Correspondence

The Oxford English Dictionary defines correspondence[2] as *"[...] close similarity, connection, or equivalence [...]"*. [9]

Definition 22 (Correspondence) *Let $R \subseteq L_1 \times L_2$ be a relation between two languages L_1 and L_2. Two artifacts a_1 and a_1 recursively correspond to each other in the sense of R if both artifacts contain parts, which correspond in the same fashion to exactly one part of the other artifact:*

$$\begin{aligned}
&(a_1, a_2) \in R \\
&\wedge \forall b_1 \in L_1 \exists! b_2 \in L_2 (b_1 \; \mathsf{partOf} \; a_1 \rightarrow (b_2 \; \mathsf{partOf} \; a_2 \wedge b_1 \; \mathsf{correspondsTo}_R \; b_2)) \\
&\wedge \forall b_2 \in L_2 \exists! b_1 \in L_1 (b_2 \; \mathsf{partOf} \; a_2 \rightarrow (b_1 \; \mathsf{partOf} \; a_2 \wedge b_2 \; \mathsf{correspondsTo}_R \; b_1)) \\
&\Rightarrow a_1 \; \mathsf{correspondsTo}_R \; a_2
\end{aligned} \tag{4.8}$$

4.3 Conformance

The Oxford English Dictionary defines conformance[3] as synonym for conformity[4] and thus in turn as *"Compliance with standards, rules, or laws"*.
[9]

Definition 23 (Conformance) *Let $L \subseteq \mathsf{Any}$ be a language, $\mathsf{Def}_L \subseteq \mathsf{Any}$ the definition language of L and $\mathsf{def}_L \in \mathsf{Def}_L$ the actual definition artifact of L. An arbitrary artifact conforms to the definition artifact of L, if and only if it is an element of L:*

$$\forall x \in \mathsf{Any}(x \; \mathsf{conformsTo} \; \mathsf{def}_L \Leftrightarrow x \in L) \tag{4.9}$$

[2]https://en.oxforddictionaries.com/definition/correspondence (retrieved 08/28/2017)
[3]https://en.oxforddictionaries.com/definition/conformance (retrieved 08/28/2017)
[4]https://en.oxforddictionaries.com/definition/conformity (retrieved 08/28/2017)

Chapter 5

Methodology

This chapter summarizes the methodology used to develop the recovery system for this thesis. In short, the recovery system is developed example-driven using the model of a fictional Human Resource Management System (HRMS) used by the 101wiki[1] for its contributions.

5.1 Example Driven Development Process

The Example Driven Development Process used to develop the recovery system for this thesis is shown in Figure 5.1. Given we acquired a suitable example corpus, which is outlined in detail in §5.2, we have to define conditions for acceptable results. These conditions can be seen as functional requirements, for instance, we want a Java class artifact to be linked via correspondence to its XML (Extensible Markup Language)-serialized form:

```
public class Company {...} correspondsTo <company>...</company>
```

Then we implement a feature which produces the specified results and run it against the example corpus. If the result meets all conditions we are done. If not, we inspect the results to check whether our predefined conditions are to vague or wrong. If so, we refine our conditions, if not, we refine our feature implementation Either way the process starts anew.

[1] https://101wiki.softlang.org/ (retrieved 12th November, 2017)

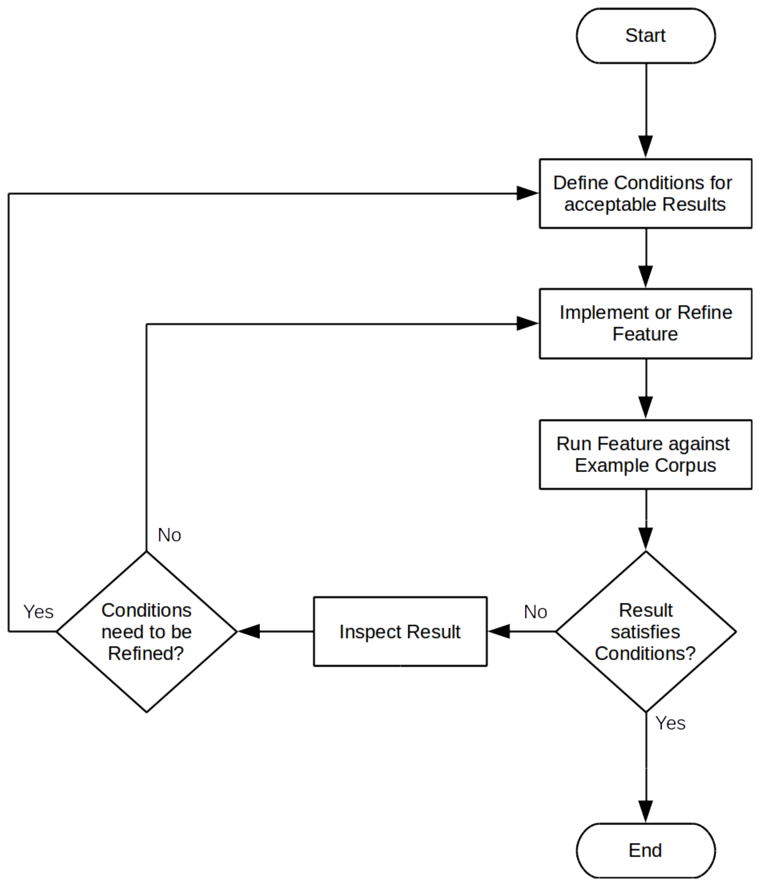

Start
Define Conditions for acceptable Results
Implement or Refine Feature
Run Feature against Example Corpus
Result satisfies Conditions?
No
Yes
Inspect Result
Conditions need to be Refined?
No
Yes
End

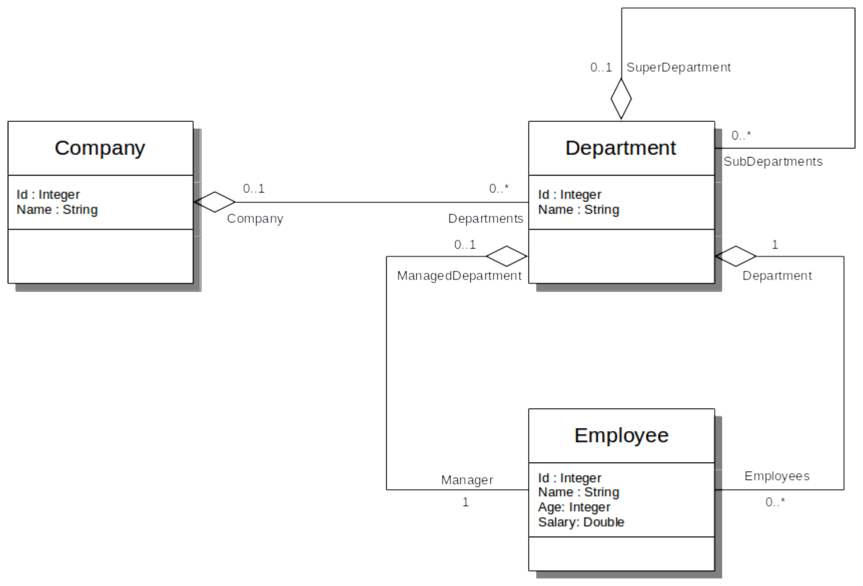
Figure 5.1 The Example Driven Development Process

5.2 Example Corpus

The example corpus used to develop the recovery system for this thesis consists of artifacts implementing a fictional HRMS within an Object-Relational- and XML-Mapping (O/R/X-Mapping) scenario using Java technologies. The model is implemented using plain Java. It is then mapped to plain XML/XSD (XML Schema Definition) with Java Architecture for XML Binding (JAXB) and to SQL/DDL statements using Hibernate mapping files and/or annotations.

5.2.1 The 101HRMS Model

101wiki Human Resource Management System (101HRMS)[2] provides a simple model of a company with many departments and employees. Figure 5.2 shows an UML (Unified Modeling Language) class diagram of a variant of this model.



This UML class diagram depicts the model of the 101HRMS. It consists of simple companies with nested departments and employees mapped to the latter.

Figure 5.2 The 101 Human Resource Management System Model

The 101HRMS model consists of companies attributed with a name. Each company accumulates departments. Each department is also attributed with a name, aggregates employees and has one employee acting as manager. Departments can further be refined into sub-departments. Each employee is attributed with a name, an age and a salary. Each entity is also attributed with an ID.

5.2.2 Linguistic Domains of the Example Corpus

The example corpus used to develop the recovery system contains artifacts implementing the 101HRMS model generated or used by Java technologies for O/R/X-Mapping, i.e. a Java model is mapped to plain XML/XSD with JAXB, to a Hi-

[2]`https://101wiki.softlang.org/101:@system` (retrieved 12th November, 2017)

bernate mapping file and to SQL (Structured Query Language)/Data Definition Language (DDL) statements. Figure 5.3 shows a schematic illustration of the linguistic domains involved:

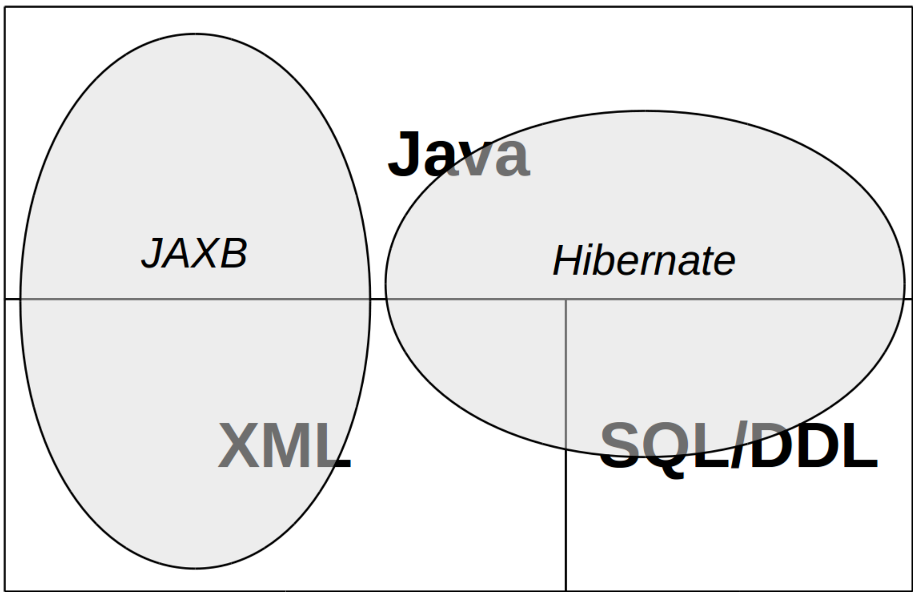
Java The language and technology used to implement the 101HRMS model.

XML The language used to serialize the 101HRMS model.

SQL/DDL The language used to persist the 101HRMS model.

JAXB The technology used to implement Object-XML-Mapping (O/X-Mapping) of the 101HRMS model.

Hibernate The technology used to implement Object-Relational-Mapping (O/R-Mapping) of the 101HRMS model.



This schematic illustration depicts the interrelation among linguistic domains of example corpus used. It depicts languages and technologies for O/R/X-Mapping with Java.

Figure 5.3 Example Corpus Domains: Java O/R/X

The languages (Java, XML & SQL) in Figure 5.3 are displayed as disjoint square sets. Technologies (JAXB & Hibernate) are displayed as oval sets intersecting languages. This is due to their linguistic nature, e.g. JAXB produces specific

Java- and XML-Code which does not necessarily intersect with code produced by other technologies. Hibernate intersects all three languages. It uses XML files or Java-Annotations for describing O/R-Mapping of a data-model and generates SQL artifacts according to that mapping. In this sense, technologies create technology-specific subsets of a languages.

Chapter 6

Requirements

This chapter summarizes functional requirements for the recovery system developed as part of this thesis.

Requirement 1 The recovery system has to recover parthood links, i.e. links capturing the relation between syntactically well-formed software artifacts and their fragments.

Requirement 2 The recovery system has to recover correspondences links, i.e. links capturing the relation between fragments of software artifacts denoting a predefined similarity among them and their fragments.

ToDo: refer to a correspondence definition

Requirement 3 The recovery system has to recover conformance links, i.e. links capturing the relation between a software artifacts or fragments denoting that one defines the other.

ToDo: refer to a conformance definition

Requirement 4 The recovery system has to run within MegaL/Xtext, i.e. the implementations of requirements Requirement 1, Requirement 2 and Requirement 3 must be compatible with the MegaL/Xtext plug-in-system.

Chapter 7

Design

This chapter summarizes the design of the recovery system (§7.1) developed for this thesis and its integration in the MegaL/Xtext environment (§7.2).

7.1 Recovery System Design

This section summarizes the design of the recovery system developed for this thesis. §7.1.1 will describe the all over process of the recovery system. §7.1.2 will describe the design core API (Application Programming Interface) and its components developed for the recovery system. §7.1.3 will describe the design of the actual system for recovering links among O/R/X-Mapping artifacts.

7.1.1 Recovery Process

The recovery process is a straight forward analysis of two artifacts. Figure 7.1 shows a flowchart depicting this. Given two artifacts as input, the recovery pro-

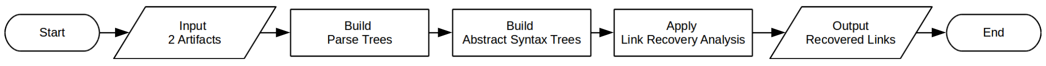


Figure 7.1 The Recovery Process

cess works as follows:

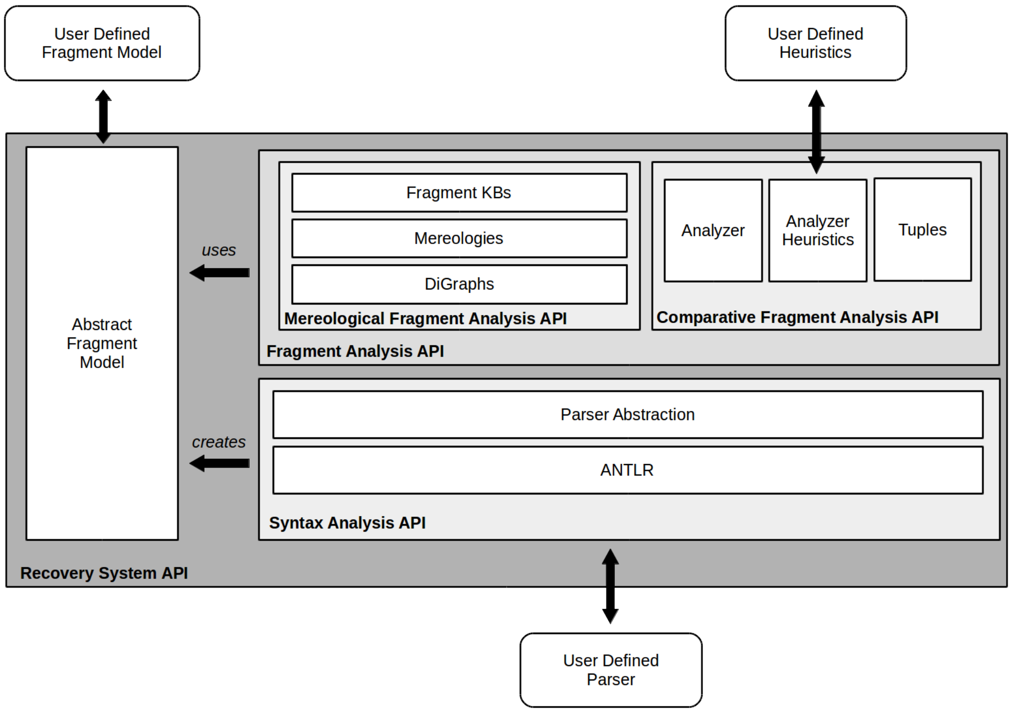
1. From each artifact a Concrete Syntax Tree (CST) or Parse Tree is constructed,

2. each Parse Tree is further refined into an Abstract Syntax Tree (AST),
3. both ASTs are compared with each other, i.e. both trees are traversed in a Depth-First Search (DFS) fashion and each pair of nodes is checked whether it can be recovered as link.

Eventually, the set of recovered links serves as output of the process.

7.1.2 Recovery API

The Recovery API is the core of the developed recovery system. It provides generalized data structures and methods for syntactic analysis of artifacts and their fragments. Figure 7.2 depicts the API as block diagram. The components of the



This block diagram depicts the functional outline of the Recovery System API (note, that this does not necessarily correspond to the package outline of its actual implementation).

Figure 7.2 The Recovery API

API are:

Abstract Fragment Model The Abstract Fragment Model serves as base model for all ASTs. It can thought of as Data Transfer Object (DTO) for the analysis components. As user of the API, i have to derive a specific AST from this model. A detailed description follows in §7.1.2.1.

Syntax Analysis API The Syntax Analysis API is the abstraction layer for parsing and AST construction. It is currently backed by ANTLR (Another Tool For Language Recognition), but its internal design is loosely coupled, so other parser libraries can be used. As user of the API, i have to implement a parser constructing an AST deriving the Abstract Fragment Model. However, if one uses ANTLR, only AST construction from a CST is required. A detailed description follows in §7.1.2.2.

Fragment Analysis API The Fragment Analysis API consists of two components:

Mereological Fragment Analysis API The Mereological Fragment Analysis API provides components for deriving parthood links between syntactically well-formed fragments. As user of the API, i only have to apply its components to constructed ASTs. A detailed description follows in §7.1.2.3.

Comparative Fragment Analysis API The Comparative Fragment Analysis API provides components for deriving links between different artifacts and their fragments. As user of the API, i have to implement one or more specialized heuristics for deciding which links can be recovered. A detailed description follows in §7.1.2.4.

7.1.2.1 Abstract Fragment Model

The Abstract Fragment Model describes a tree in which each node represents an syntactically well-formed fragment. Figure 7.3 shows an UML class diagram of the model. The resulting tree data structure is doubly-linked, i.e. an `IFragment` node aggregates references to its children an to its parent, given it is not the root node. Each fragment `IFragment` contains the text it represents. In order to distinguish fragments which represent the same text, each node also carries the text's position in the artifact through `IFragmentPosition` instances. Positions inside the text are determined by the start and ending line number as well as the corre-

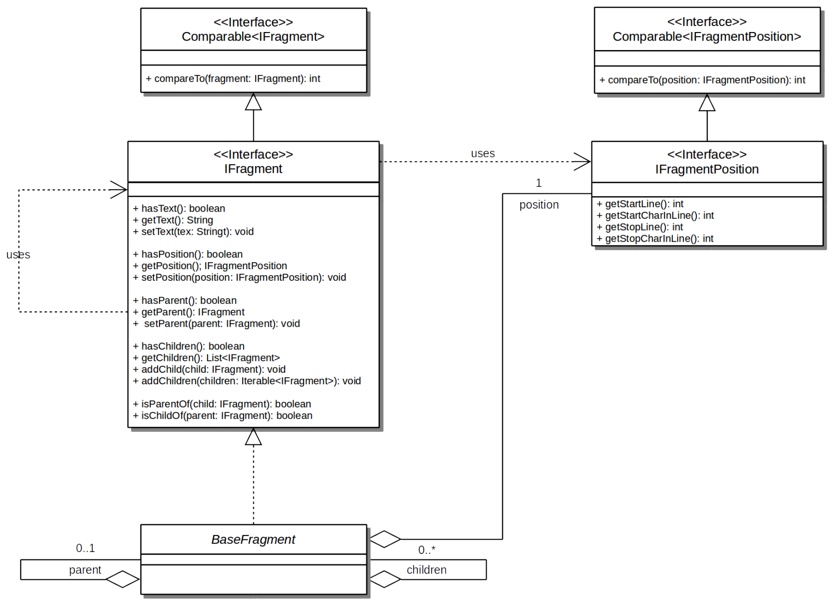

<<Interface>>
Comparable<IFragment>
+ compareTo(fragment: IFragment): int
<<Interface>>
Comparable<IFragmentPosition>
+ compareTo(position: IFragmentPosition): int
<<Interface>>
IFragment
+ hasText(): boolean
+ getText(): String
+ setText(tex: Stringt): void
+ hasPosition(): boolean
+ getPosition(); IFragmentPosition
+ setPosition(position: IFragmentPosition): void
+ hasParent(): boolean
+ getParent(): IFragment
+ setParent(parent: IFragment): void
+ hasChildren(): boolean
+ getChildren(): List<IFragment>
+ addChild(child: IFragment): void
+ addChildren(children: Iterable<IFragment>): void
+ isParentOf(child: IFragment): boolean
+ isChildOf(parent: IFragment): boolean
uses
uses
1
position
<<Interface>>
IFragmentPosition
+ getStartLine(): int
+ getStartCharInLine(): int
+ getStopLine(): int
+ getStopCharInLine(): int
BaseFragment
0..1
parent
0..*
children

Figure 7.3 The Abstract Fragment Model

sponding first and last character inside the line. Most of `IFragment`'s relevant code for trees is pre-implemented in the `BaseFragment` abstract class.

7.1.2.2 Syntax Analysis API

The Syntax Analysis API provides abstraction for AST construction. The API itself is really small, it only consists of the `IParser` and `IParserFactory` interfaces. However, its implementation for ANTLR is designed to reduce ANTLR-specific boilerplate code. Figure 7.4 shows the UML class diagram for the ANTLR backed implementation. One can see, that this implementation makes heavy use of the Abstract Factory Pattern [4]. This allows for a quick and easy definition of new parsers as shown in Figure 7.5.

The other advantage of the Abstract Factory Pattern is that the instantiation of all relevant classes necessary for the creation of ANTLR Parse Trees takes place in the predefined `AntlrParser` class. This is exemplified by Figure 7.6.

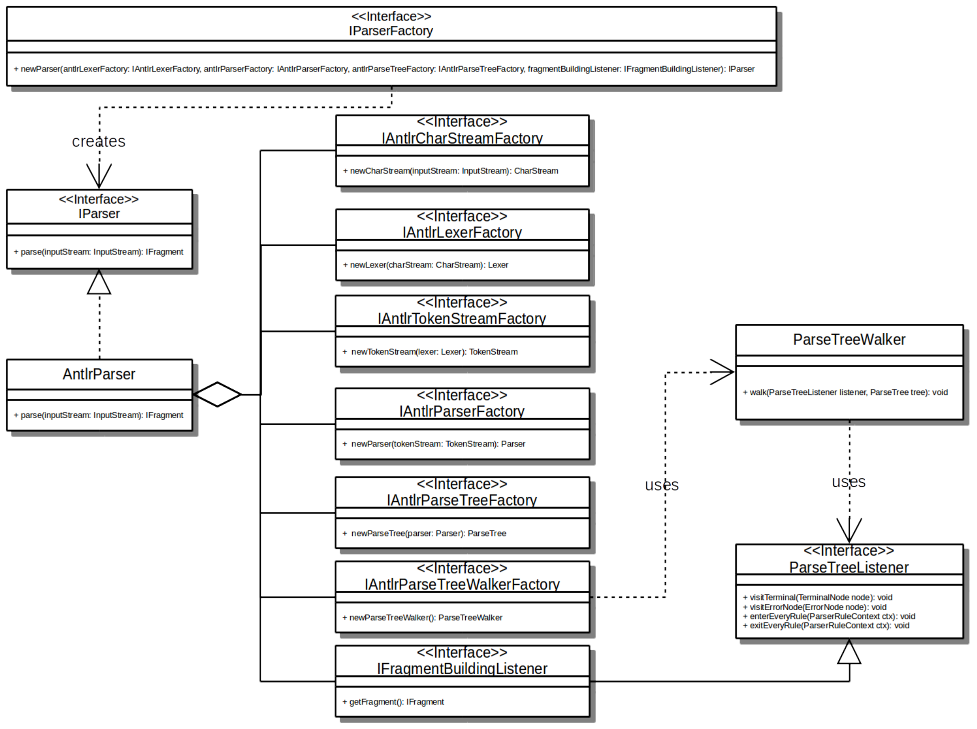

<<Interface>>
IParserFactory
+ newParser(antlrLexerFactory: IAntlrLexerFactory, antlrParserFactory: IAntlrParserFactory, antlrParseTreeFactory: IAntlrParseTreeFactory, fragmentBuildingListener: IFragmentBuildingListener): IParser
creates
<<Interface>>
IParser
+ parse(inputStream: InputStream): IFragment
AntlrParser
+ parse(inputStream: InputStream): IFragment
<<Interface>>
IAntlrCharStreamFactory
+ newCharStream(inputStream: InputStream): CharStream
<<Interface>>
IAntlrLexerFactory
+ newLexer(charStream: CharStream): Lexer
<<Interface>>
IAntlrTokenStreamFactory
+ newTokenStream(lexer: Lexer): TokenStream
<<Interface>>
IAntlrParserFactory
+ newParser(tokenStream: TokenStream): Parser
<<Interface>>
IAntlrParseTreeFactory
+ newParseTree(parser: Parser): ParseTree
<<Interface>>
IAntlrParseTreeWalkerFactory
+ newParseTreeWalker(): ParseTreeWalker
<<Interface>>
IFragmentBuildingListener
+ getFragment(): IFragment
ParseTreeWalker
+ walk(ParseTreeListener listener, ParseTree tree): void
uses
uses
<<Interface>>
ParseTreeListener
+ visitTerminal(TerminalNode node): void
+ visitErrorNode(ErrorNode node): void
+ enterEveryRule(ParserRuleContext ctx): void
+ exitEveryRule(ParserRuleContext ctx): void

Figure 7.4 The Syntax Analysis API

```
...
IParser java8Parser = parserFactory.newParser(
    Java8Lexer::new,
    Java8Parser::new,
    Java8Parser::compilationUnit,
    Java8FragmentBuildingListener::new);
...
```

This example demonstrates the creation of a new parser using an `IParserFactory` instance with Java8 features. Lexer and Parser classes are generated by ANTLR. A suitable listener has to be implemented.

Figure 7.5 `IParserFactory` Usage Example

```
...
CharStream charStream = antlrCharStreamFactory.newCharStream(inputStream);
Lexer lexer = antlrLexerFactory.newLexer(charStream);
TokenStream tokenStream = antlrTokenStreamFactory.newTokenStream(lexer);
Parser parser = antlrParserFactory.newParser(tokenStream);
ParseTree parseTree = antlrParseTreeFactory.newParseTree(parser);
...
```

This example demonstrates the creation of an ANTLR `ParseTree` instance as implemented by the `AntlrParser` class.

Figure 7.6 `AntlrParser` Parse Tree Creation

Creation of ASTs or fragment trees is done using the `ParseTreeWalker` and `ParseTreeListener` infrastructure provided by ANTLR [11]. This is an variation of the Observer Pattern [4]. Listeners in conjunction with walkers are an alternative to the Visitor Pattern [4] provided by ANTLR. An instance of `ParseTreeWalker` traverses a Parse Tree using DFS. During traversal, a designated method of `ParseTreeListener` is executed. For AST creation, an API user has to implement the `IFragmentBuildingListener` interface, which is an extension of the `ParseTreeListener` interface. The creation of a fragment tree by `AntlrParser` is exemplified in Figure 7.7.

```
...
ParseTreeWalker parseTreeWalker = antlrParseTreeWalkerFactory.newParseTreeWalker();
parseTreeWalker.walk(fragmentBuildingListener, parseTree);
IFragment fragment = fragmentBuildingListener.getFragment();
...
```

This example demonstrates the creation of an `IFragment` AST instance as implemented by the `AntlrParser` class.

Figure 7.7 `AntlrParser` Fragment Creation

7.1.2.3 Mereological Fragment Analysis API

The Mereological Fragment Analysis API provides components for deriving parthood links between syntactically well-formed fragments. Direct parthood links are recovered from parent/child relationships within an `IFragment` AST. However, because parthood is transitive, we also need to recover these links. This is done using a digraph data structure upon which we can compute its reflexive transitive closure.

Figure 7.8 shows an UML class diagram depicting the relevant classes and interfaces of the Mereological Fragment Analysis API. At its heart, the API utilizes the generic `IDiGraph` interface whose implementations provide means for interrelating comparable objects, i.e. `IFragment` instances.

This digraph is wrapped by generic `IMereology` implementations, a data structure providing query methods on the topic of mereology, i.e. parthood relations. Mereologys are constructed using the Builder Pattern [4]. An `IMereologyBuilder` instance adds nodes and edges into its digraph with semantically named methods allowing a descriptive programming style.

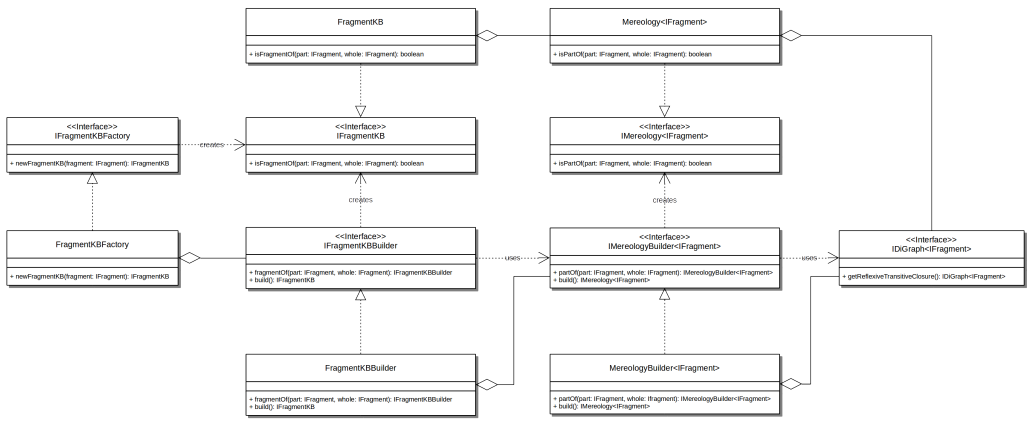

FragmentKB
+ isFragmentOf(part: IFragment, whole: IFragment): boolean
Mereology<IFragment>
+ isPartOf(part: IFragment, whole: IFragment): boolean
<<Interface>>
IFragmentKBFactory
+ newFragmentKB(fragment: IFragment): IFragmentKB
creates
<<Interface>>
IFragmentKB
+ isFragmentOf(part: IFragment, whole: IFragment): boolean
<<Interface>>
IMereology<IFragment>
+ isPartOf(part: IFragment, whole: IFragment): boolean
creates
creates
FragmentKBFactory
+ newFragmentKB(fragment: IFragment): IFragmentKB
<<Interface>>
IFragmentKBBuilder
+ fragmentOf(part: IFragment, whole: IFragment): IFragmentKBBuilder
+ build(): IFragmentKB
uses
<<Interface>>
IMereologyBuilder<IFragment>
+ partOf(part: IFragment, whole: IFragment): IMereologyBuilder<IFragment>
+ build(): IMereology<IFragment>
uses
<<Interface>>
IDiGraph<IFragment>
+ getReflexiveTransitiveClosure(): IDiGraph<IFragment>
FragmentKBBuilder
+ fragmentOf(part: IFragment, whole: IFragment): IFragmentKBBuilder
+ build(): IFragmentKB
MereologyBuilder<IFragment>
+ partOf(part: IFragment, whole: Ifragment): IMereologyBuilder<IFragment>
+ build(): IMereology<IFragment>

Figure 7.8 Mereological Fragment Analysis API

`IMereology`'s are then further wrapped by `IFragmentKB` (a Knowledge Base over `IFragment`) implementations in order to avoid dealing with generics throughout the system. This is also done utilizing the Builder Pattern.

The recovery of parthood links is encapsulated through `IFragmentKBFactory` using the Abstract Factory Pattern, which computes `IFragmentKB` instances from an `IFragment` AST as input.

7.1.2.4 Comparative Fragment Analysis API

The Comparative Fragment Analysis API provides components for deriving links from two `IFragment` ASTs generated from different artifacts. Figure 7.9 shows an UML class diagram depicting the relevant interfaces and classes of the API. It utilizes the Strategy Pattern [4] through the `IFragmentAnalyzerHeuristic` interface since concrete behavior of the recovery analysis is for the user to implement. In this context, strategies are called heuristics, because recovery of links is not necessarily based on optimal, i.e. absolutely correct, solutions. Instead strategies may also implement practical solutions with reasonably sufficient results.

`IFragmentAnalyzer` objects allow one to apply multiple heuristics, since there may be more than one practical approach to achieve a goal. Also this allows strategies to keep a relatively small implementation footprint.

Recovered links are captured and stored in an `IBinaryRelation` instance which works like a set of pairs.

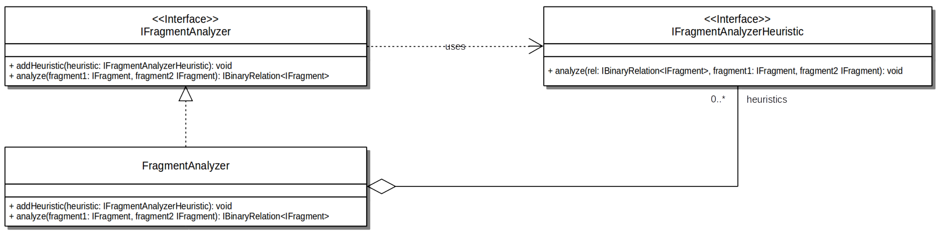
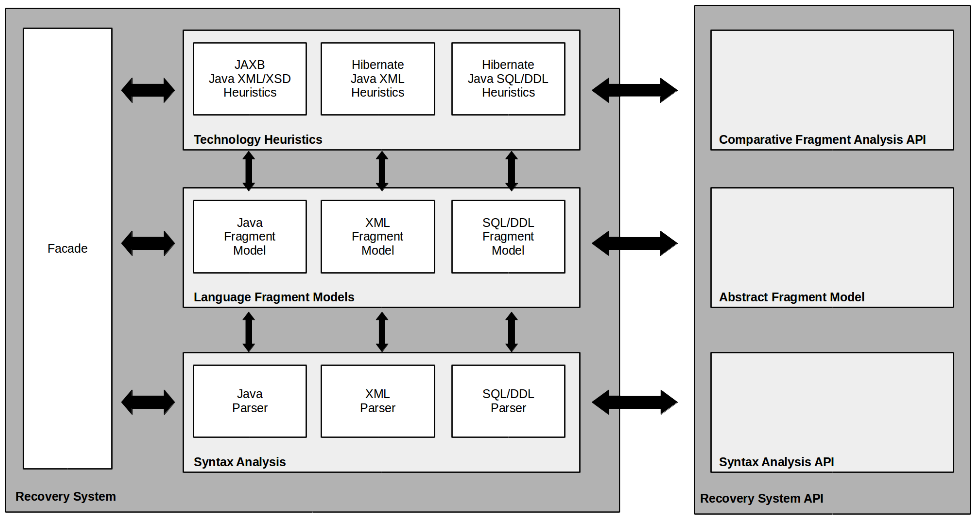

<<Interface>>
IFragmentAnalyzer
+ addHeuristic(heuristic: IFragmentAnalyzerHeuristic): void
+ analyze(fragment1: IFragment, fragment2 IFragment): IBinaryRelation<IFragment>
uses
<<Interface>>
IFragmentAnalyzerHeuristic
+ analyze(rel: IBinaryRelation<IFragment>, fragment1: IFragment, fragment2 IFragment): void
0..*
heuristics
FragmentAnalyzer
+ addHeuristic(heuristic: IFragmentAnalyzerHeuristic): void
+ analyze(fragment1: IFragment, fragment2 IFragment): IBinaryRelation<IFragment>

Figure 7.9 Comparative Fragment Analysis API

7.1.3 Recovery System

The Recovery System is the actual implementation of parthood, correspondence and conformance link recovery for Java based O/R/X-Mapping artifacts. Figure 7.10 shows a block diagram outlining the functional components of the system.



This block diagram depicts the functional outline of the Recovery System and its dependencies to the Recovery System API (note, that this does not necessarily correspond to the package outline of its actual implementation).

Figure 7.10 The Recovery System

The Recovery System utilizes the Syntactic Analysis API described in §7.1.2.2 for Java, XML and SQL/DDL. For this, the Abstract Fragment Model described in §7.1.2.1 is implemented. Note, fragment models do not implement AST suitable for compilation of the targeted language. Syntactic features unnecessary for the intended analysis, like local variable declarations, arithmetic expressions or invocations, are omitted. Fragment models rather focuses on structural features of the languages at hand.

The Comparative Fragment Analysis API described in §7.1.2.4 is implemented with focus on artifacts of technologies, namely JAXB and Hibernate. It implements heuristics for link recovery between:

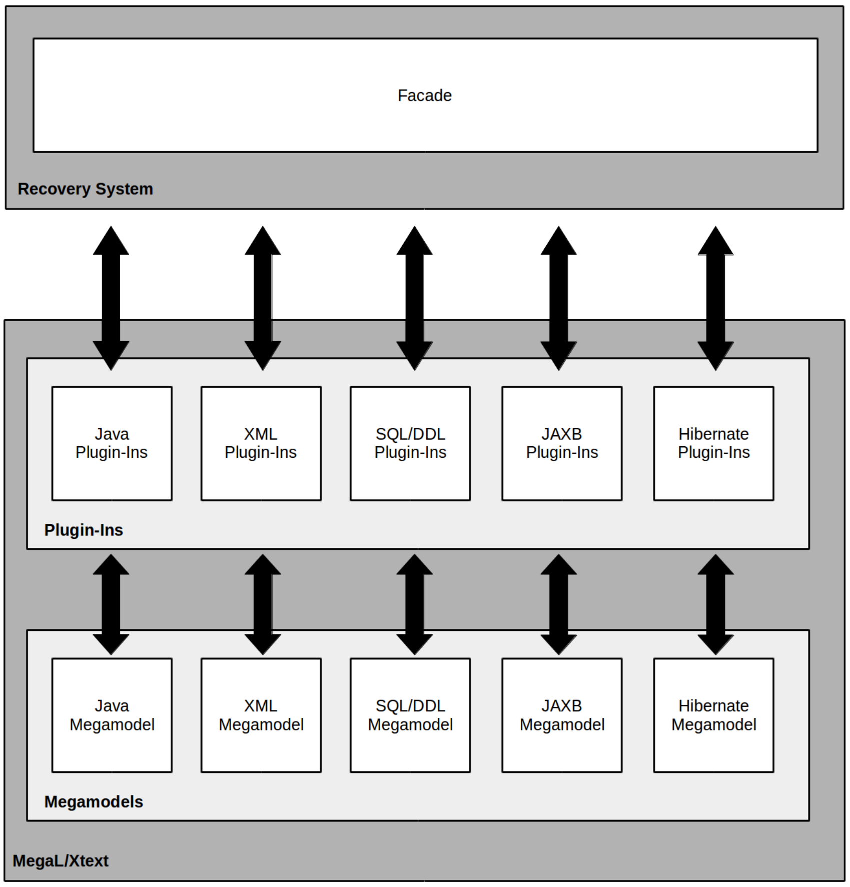
- Java and XML for JAXB artifacts, i.e. Java models are serialized as XML
- Java and XSD for JAXB artifacts, i.e. Java models are serialized as XSD
- XML and XSD for JAXB artifacts, i.e. the two previous scenarios occurred
- Java and XML for Hibernate mapping artifacts, i.e. Hibernate uses XML meta-data for O/R-Mapping.
- Java and SQL/DDL for Hibernate generated SQL artifacts, i.e. Hibernate uses Java annotations for O/R-Mapping

On top of the actual implementation, the Recovery System also utilizes the Facade Pattern [4]. This is to provide a single access point for all implemented analysis features.

7.2 Megal/Xtext Integration Design

This section summarizes the design of the recovery system's integration in the MegaL/Xtext environment. Figure 7.11 shows a block diagram depicting the outline this integration. Note, the Recovery System is implemented as separate project, which is then included as reference (through a `.jar` file) in a MegaL/Xtext project.

MegaL/Xtext provides a plug-in system, which allows to bind special Java classes to `Plugin` entities. Code of such classes is then executed during the evaluation of a megamodel. **ToDo: needs reference** Figure 7.12 exemplifies the declaration of Plug-in withing MegaL.



This block diagram depicts the functional outline of the Recovery System's integration into the MegaL/Xtext environment (note, that this does not necessarily correspond to the outline of its actual implementation).

Figure 7.11 Integration of the Recovery System int MegaL/Xtext

```
...
JavaFragmentRecoveryPlugin : Plugin
JavaFragmentRecoveryPlugin realizationOf Java
JavaFragmentRecoveryPlugin partOf FileFragmentRecoveryReasonerPlugin
JavaFragmentRecoveryPlugin = 'classpath:org.softlang.megal.plugins.impl.java.JavaFragmentRecoveryPlugin'
...
```

This snippet demonstrates the instantiation of `JavaFragmentRecoveryPlugin` as part of `FileFragmentRecoveryReasonerPlugin` in MegaL.

Figure 7.12 MegaL Plug-In Instantiation

The Recovery System or rather its facade (see §7.1.3) is used to implement MegaL plug-ins. These Plug-ins are then in turn bound within megamodels, which are divided by language and technology, i.e. there are separate MegaL modules for Java, XML, SQL, Hibernate and JAXB.

Chapter 8

Implementation

This chapter summarizes the implementation of crucial parts of recovery system implemented for this thesis. §8.1 covers the implementation of fragment recovery. §8.2 covers the implementation of parthood link recovery. §8.3 covers the implementation of correspondence and conformance link recovery

8.1 Recovering Fragments

This section summarizes the implementation of fragment recovery. Fragments are syntactically well-formed pieces of code. For instance, consider the following Java class:

```java
public class Company {
    ...
    private String name;
    ...
    public String getName() {
        return name;
    }
    ...
}
```

It consists of the following fragments among others:

- a field declaration fragment for `name`:

  ```java
  private String name;
  ```

- a method declaration fragment for an accessor-method of `name`:

  ```java
  public String getName() {
      return name;
  }
  ```

- the return statement of the accessor-method:

```
return name;
```

- the class declaration itself can also be considered a fragment.

The task of recovering fragments is to add entities for each of such code pieces into a megamodel. Figure 8.1 exemplifies the recovery of Java fragments in an idealized form.

1. a Parse Tree is generated from an artifact; this is done via ANTLR as described in §7.1.2.2 and does not require further explanation, since it is only an application of the most common ANTLR use case
2. then the Parse Tree is transformed into an concrete fragment AST model; this is also done relying on ANTLR tree traversal utilities and described in detail in §8.1.1
3. eventually, nodes of the generated fragment AST are added to a megamodel as entities; the details are described in §8.1.2.

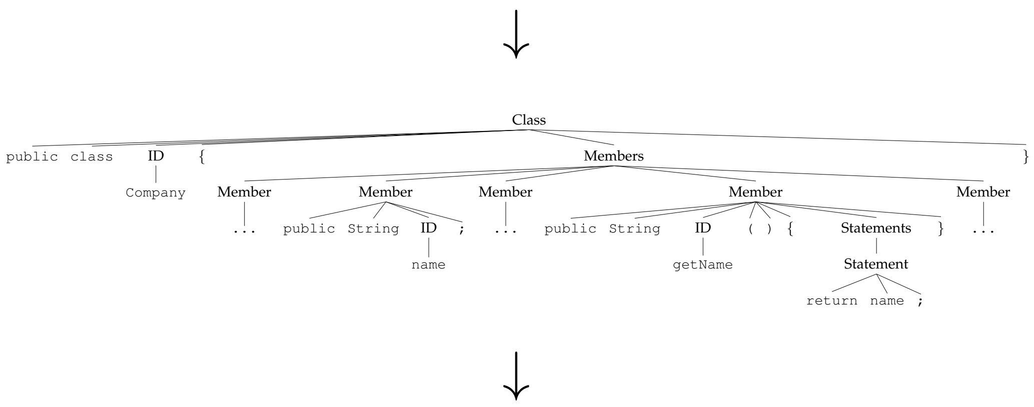
Note, fragment Recovery for XML and SQL/DDL is implemented in a similar fashion. If there is a noteworthy difference for other languages it will be explored, otherwise we keep using Java as example domain for the following sections of §8.1.

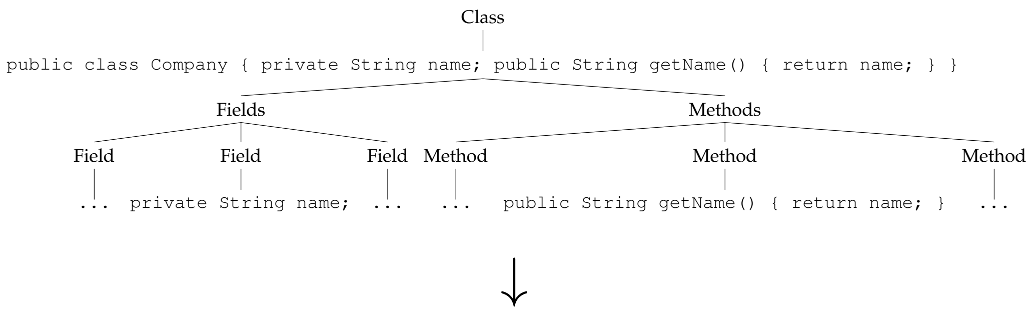
8.1.1 Concrete Fragment AST Models

As mentioned in §7.1.2.1, Concrete Fragment AST Models are derivations of the Abstract Fragment Model of the Recovery System API. Figure 8.2 shows an UML class diagram of the implemented Fragment AST for Java. All fragment classes derive from IFragment through the base class `JavaFragment`. From here, several specializations are introduced:

- `IdentfiedJavaFragment` for constructs with identifiers
- `ModifiedJavaFragment` for constructs with modifiers like `private`, `public`, `final`, `abstract`, `static`, etc. or annotation meta-data
- `TypedJavaFragment` for constructs with distinct (return-) type like fields, methods or variables

```
public class Company {
   ...
   private String name;
   ...
   public String getName() {
      return name;
   }
   ...
}
```




Class
public class Company { private String name; public String getName() { return name; } }
Fields
Field
...
Field
private String name;
Field
...
Methods
Method
...
Method
public String getName() { return name; }
Method
...

```
Company : Fragment
Company elementOf JavaClassFragments
...
Company.name : Fragment
Company.name elementOf JavaFieldFragments
...
Company.getName : Fragment
Company.getName elementOf JavaMethodFragments
```

This picture shows an idealized recovery of Java fragments:

code artifact → Parse Tree → fragment AST → megamodel

Both Parse Tree and AST are depicted in a simplified, schematic form.

Figure 8.1 Idealized Recovery of Java Fragments

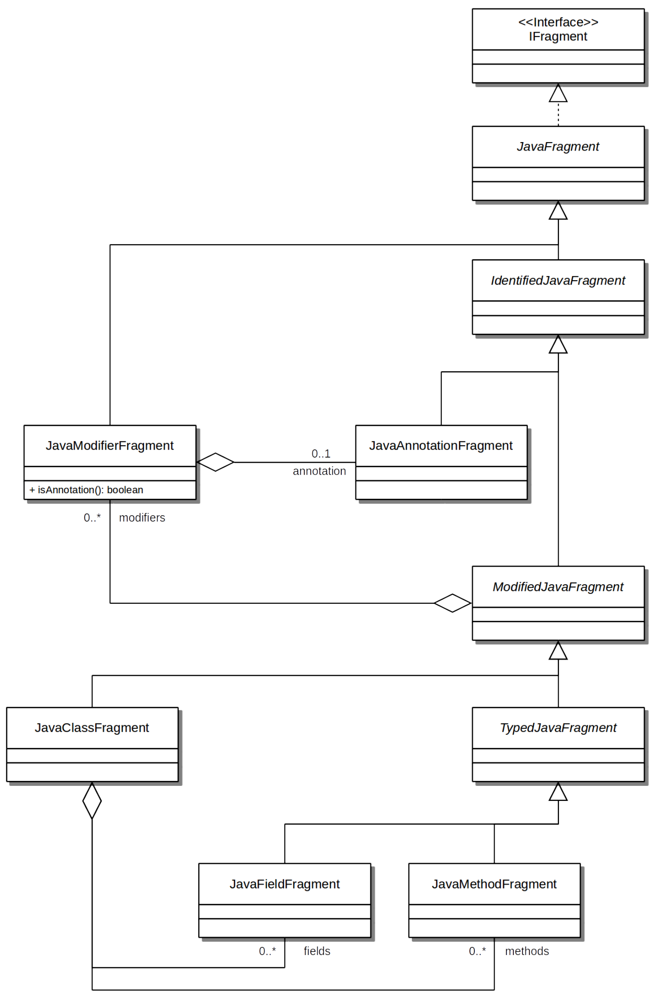


Figure 8.2 Java Fragment AST Model

Note, the fragment model for Java only focuses on structural syntactical features of the language. Everything below method signature level is omitted, because fragments of this level are not important for the scope of this thesis. On the other hand, annotations are captured, since they provide significant information for further recovery analysis. Hibernate and JAXB rely on annotations for O/R/X-Mapping.

The AST is constructed using ANTLR `ParseTreeListener` approach as described in §7.1.2.2. Figure 8.3 shows an excerpt from the listener implementation used to construct the the AST for Java fragments. Here, fragment instances are

```
...
@Override
public void exitMethodDeclaration(Java8Parser.MethodDeclarationContext ctx) {
    javaMethodFragments.push(java8FragmentFactory.newJavaMethodFragment(ctx, javaMethodModifierFragments));
}

@Override
public void exitNormalClassDeclaration(Java8Parser.NormalClassDeclarationContext ctx) {
    javaClassFragments.push(java8FragmentFactory.newJavaClassFragment(ctx, javaClassModifierFragments,
            javaFieldFragments, javaMethodFragments, declaredPackage));
}
...
```

This snippet shows an excerpt of the listener used to construct Java fragment ASTs.

Figure 8.3 Construction of Java Fragment ASTs

created when the listener leaves a certain node of the Parse Tree. Creation is delegated to a factory. The listener pushes fragments onto stacks, making the overall implementation work like a pushdown automaton or stack machine.

8.1.2 Megamodeling Fragments

asdf

```
...
Language < Set
...
fragmentLanguageOf < Language * Language
...
Java : Language

JavaFragments : Language
JavaFragments fragmentLanguageOf Java

JavaClassFragments : Language
JavaClassFragments subsetOf JavaFragments

JavaFieldFragments : Language
JavaFieldFragments subsetOf JavaFragments

JavaMethodFragments : Language
JavaMethodFragments subsetOf JavaFragments
...
```

Figure 8.4 A Megamodel for Java Fragments

8.2 Recovering Parthood Links

This section summarizes the implementation of parthood link recovery.

8.3 Recovering Correspondence & Conformance Links

This section the implementation of correspondence and conformance link recovery

Chapter 9

Results

TBD.

Chapter 10

Conclusion

TBD.

Glossary

101HRMS 101wiki[1] Human Resource Management System[2]. The model used by the 101wiki for its contributions. 31, 32, *see also* HRMS

Abstract Factory Pattern A creational GoF (Gang of Four) pattern used in software design to decouple instantiation from usage of objects. Hides the concrete nature of created instances. 38, 41

ANTLR Another Tool For Language Recognition. 37–40

API Application Programming Interface. 35–38, 40–43, 51

AST Abstract Syntax Tree: A tree data structure representing the abstract syntax of a parsed text. This tree omits syntactic features like parentheses for grouping or semicolons for sequencing. 36–38, 40, 41, 43

Builder Pattern A creational GoF pattern used in software design to prevent constructor parameters from piling up. 40, 41

conformance The relation between . 42, 46, 47

correspondence The relation between . 42, 46, 47

CST Concrete Syntax Tree: A tree data structure representing the concrete syntax of a parsed text.. 35, 37

DDL Data Definition Language. Language or subset of a language used to describe structure and content of data. 32

[1]https://101wiki.softlang.org/ (retrieved 12th November, 2017)
[2]https://101wiki.softlang.org/101:@system (retrieved 12th November, 2017)

DFS The algorithmic concept of traversing a tree or graph data structure 'top-down' until reaching the end of a path before backtracking and traversing another path. 36, 40

DTO Data Transfer Object. Objects with no relevant (business) logic of their own. Their sole purpose is to carry data between layers of a software system. 37

Facade Pattern A structural GoF pattern used in software design to simplify the usage of complex systems or APIs. It provides single access point for such system. Such access points are called facades. 43

fragment A syntactically well-formed piece of a possibly larger text. v, 46

GoF Gang of Four. A group of authors (Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides) publishing on the subject of object-oriented software design. The term may also refer to design patterns described in their book *Design Patterns: Elements of Reusable Object-Oriented Software* [4]. 50–52

Hibernate The Hibernate ORM Framework. 30–33, 43, 45

HRMS Human Resource Management System. 29, 30

Java The Java Programming Language and Platform. v, 29–33, 39, 43, 45, 46

JAXB Java Architecture for XML Binding. 30–32, 43, 45

MegaL The megamodeling language developed by the Softlang Team at the University of Koblenz-Landau for descriptively and prescriptively modeling linguistic architectures of software systems. 43–45, 51

MegaL/Xtext The Xtext implementation and eclipse IDE integration of MegaL. 34, 35, 43, 44

megamodel . v, 43, 44, 46

mereology . 40

O/R-Mapping Object-Relational-Mapping. 32, 33, 43

O/R/X-Mapping Object-Relational- and XML-Mapping. 30–32, 35, 42

O/X-Mapping Object-XML-Mapping. 32

Observer Pattern A behavioral GoF pattern used in software design to propagate state changes from one object to many dependent objects. 40

ORM . 51, *see* O/R-Mapping

Parse Tree . iv, 35, 36, 38–40, *see* CST

parthood The relation between an entity and its constituent parts. 40–42, 46, 47

SQL Structured Query Language. 32, 33, 43, 45

SQL/DDL The DDL subset of SQL. 30, 43

Strategy Pattern A behavioral GoF pattern used in software design to separate behavior from structure. It allows to encapsulate and reuse behavior as part of the configuration of larger constructs. 41

UML Unified Modeling Language. 31, 37, 38, 40, 41

Visitor Pattern A behavioral GoF pattern used in software design to separate behavior from structure. Visitors facilitate the extension of behavior without modifying structure. The Visitor Pattern can be used to traverse object graphs. 40

XML Extensible Markup Language. 29–33, 43, 45

XSD XML Schema Definition. 30, 31, 43

Bibliography

[1] Anya Helene Bagge and Vadim Zaytsev. “Languages, Models and Megamodels”. In: *Post-proceedings of the Seventh Seminar on Advanced Techniques and Tools for Software Evolution, SATToSE 2014, L'Aquila, Italy, 9-11 July 2014.* 2014, pp. 132–143. URL: http://ceur-ws.org/Vol-1354/paper-12.pdf.

[2] Jean-Marie Favre, Ralf Lämmel, and Andrei Varanovich. “Modeling the Linguistic Architecture of Software Products”. In: *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings*. 2012, pp. 151–167. DOI: 10.1007/978-3-642-33666-9_11. URL: http://dx.doi.org/10.1007/978-3-642-33666-9_11.

[3] Jean-Marie Favre and Tam Nguyen. “Towards a Megamodel to Model Software Evolution Through Transformations”. In: *Electr. Notes Theor. Comput. Sci.* 127.3 (2005), pp. 59–74. DOI: 10.1016/j.entcs.2004.08.034. URL: http://dx.doi.org/10.1016/j.entcs.2004.08.034.

[4] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[5] Orlena Gotel et al. “Traceability Fundamentals”. In: *Software and Systems Traceability.* 2012, pp. 3–22. DOI: 10.1007/978-1-4471-2239-5_1. URL: https://doi.org/10.1007/978-1-4471-2239-5_1.

[6] Nicola Guarino, Daniel Oberle, and Steffen Staab. "What Is an Ontology?" In: *Handbook on Ontologies*. 2009, pp. 1–17. DOI: 10.1007/978-3-540-92673-3_0. URL: https://doi.org/10.1007/978-3-540-92673-3_0.

[7] Lukas Härtel. "Linguistic architecture on the workbench". Bachelor Thesis. University of Koblenz-Landau, Oct. 2015.

[8] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (Dec. 1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=2238.

[9] Ralf Lämmel. "Coupled software transformations revisited". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam, The Netherlands, October 31 - November 1, 2016*. 2016, pp. 239–252. URL: http://dl.acm.org/citation.cfm?id=2997366.

[10] Ralf Lämmel and Andrei Varanovich. "Interpretation of Linguistic Architecture". In: *Modelling Foundations and Applications - 10th European Conference, ECMFA 2014, Held as Part of STAF 2014, York, UK, July 21-25, 2014. Proceedings*. 2014, pp. 67–82. DOI: 10.1007/978-3-319-09195-2_5. URL: http://dx.doi.org/10.1007/978-3-319-09195-2_5.

[11] Terence Parr. *The Definitive ANTLR 4 Reference*. 2nd. Pragmatic Bookshelf, 2013. ISBN: 1934356999, 9781934356999.

[12] Alan Rector et al. *Simple part-whole relations in OWL Ontologies*. retrieved 12. June 2017. Aug. 2005. URL: http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/simple-part-whole-relations-v1.5.html.

[13] Gunther Schmidt and Thomas Ströhlein. *Relationen und Graphen*. Mathematik für Informatiker. Springer, 1989. ISBN: 3-540-50304-8.

[14] Uwe Schöning. *Theoretische Informatik - kurzgefaßt (3. Aufl.)* Hochschultaschenbuch. Spektrum Akademischer Verlag, 1997. ISBN: 978-3-8274-0250-9.

[15] Achille C. Varzi. "Mereology". In: *The Stanford encyclopedia of philosophy*. Ed. by Edward N. Zalta. Winter 2016 ed. retrieved 27. July 2017. 2016. URL: https://plato.stanford.edu/archives/win2016/entries/mereology/.

[16] Achille C. Varzi. "Parts, Wholes, and Part-Whole Relations: The Prospects of Mereotopology". In: *Data Knowl. Eng.* 20.3 (1996), pp. 259–286. DOI: 10.1016/S0169-023X(96)00017-1. URL: http://dx.doi.org/10.1016/S0169-023X(96)00017-1.

[17] Stefan Winkler and Jens Pilgrim. "A Survey of Traceability in Requirements Engineering and Model-driven Development". In: *Softw. Syst. Model.* 9.4 (Sept. 2010), pp. 529–565. ISSN: 1619-1366. DOI: 10.1007/s10270-009-0145-0. URL: http://dx.doi.org/10.1007/s10270-009-0145-0.