

Contents

1	Introduction	2
2	Environment Configuration	2
2.1	Eclipse IDE and other required tools	2
2.2	Project dependencies	3
2.2.1	The RCA metamodel project	3
2.2.2	The UML2RCA project	3
2.3	Creating a new project using UML2RCA and RCA	3
2.3.1	Notes	4
3	FCA (Formal Context Analysis)	4
3.1	Introduction	4
3.2	Formal contexts	4
3.3	Formal concepts	5
3.4	Subsumption relation	5
3.5	Concept lattices	7
3.6	Attribute/Object-introducing and neutral concepts	7
3.7	OC-posets, AC-posets and AOC-posets	7
4	RCA (Relational Context Analysis)	10
4.1	Introduction	10
4.2	Relational contexts	10
4.2.1	Note	10
4.3	RCFs (Relational Context Families)	10
4.4	Relational Scaling	12
4.4.1	Scaling operators & relational attributes	12
4.4.2	Types of scaling operators	13
4.4.3	Properties of relational scaling	13
4.4.4	Example of relational scaling: The existential scaling operator	13
4.5	RCA iterative algorithm	14
5	The RCA Metamodel	14
5.1	The Core viewpoint	17
5.1.1	Description	17
5.2	The Formal Context viewpoint	17
5.2.1	Description	17
5.3	The Relation Context viewpoint	17
5.3.1	Description	17
5.4	The Global viewpoint	21
6	The Generic Metamodel Transformation Architecture	21
6.1	Introduction	21
6.2	Transformations, Transformation strategies, and Transformers	21

6.3	Adaptations, Adaptation strategies, and Adapters	21
6.4	Conversions, Conversion strategies, and Converters	21
6.5	Model Management	21
6.6	Conflict Management	21
7	The UML-to-RCA use-case	21
7.1	Introduction	21
7.2	Model Management	21
7.2.1	Ecore Model Manager	21
7.2.2	Ecore Model State	21
7.3	Adaptations (X-UML to R-UML)	23
7.3.1	Associations & Dependencies	23
7.3.2	Generalizations	23
7.4	Conversions (R-UML to unpopulated RCFs)	23
7.4.1	Associations to Relational Contexts	24
7.4.2	Concrete Classes to Formal Contexts	24
7.4.3	Boolean Attributes to Formal Attributes	24
7.5	Instance migrations (populating the RCFs)	24

1 Introduction

This project defines UML-to-RCA metamodel transformations. It introduces a generic metamodel transformation architecture and specializes it into the UML-to-RCA use-case. It then uses data provided by the Knowmana project to validate both the architecture and the defined transformations. It relies on the RCA metamodel (*as provided by the rca project available at <https://github.com/anonbnr/rca>*) and the UML2 metamodel (*as provided by the `org.eclipse.uml2.uml` package*).

2 Environment Configuration

2.1 Eclipse IDE and other required tools

1. install Eclipse IDE pre-bundled with the Eclipse Modeling Tools: <https://www.eclipse.org/downloads/packages/release/2020-03/r/eclipse-modeling-tools>
2. install Apache Maven to automatically handle dependencies, as follows: Help -> Install New Software -> add a repository named “**m2eclipse**” at “<http://download.eclipse.org/technology/m2e/releases>” -> install Maven Integration for Eclipse -> Restart Eclipse
3. install the Papyrus project must to create, visualize, import and export the .uml model files, instead of using the Ecore Modeling Editor, as follows: Help -> Eclipse Marketplace -> search for Papyrus Software Designer in All Available Markets -> install Papyrus Software Designer 1.1.0 -> Restart Eclipse.

2.2 Project dependencies

2.2.1 The RCA metamodel project

The RCA metamodel project must be imported into the workspace as follows:

1. Window -> Perspective -> Open Perspective -> Other... -> Choose Git -> right click on the Git repositories perspective -> Clone a Git Repository... -> copy `https://github.com/anonbnr/rca.git` into the URI field -> Next -> Enter GitHub credentials -> Next -> Select the proper directory to import the project -> Finish -> Enter GitHub credentials
2. Window -> Perspective -> Open Perspective -> Java -> right click on the Package Explorer perspective -> Import... -> Git -> Projects from Git -> Next -> Existing local repository -> rca -> Next -> Next -> Finish

2.2.2 The UML2RCA project

The UML2RCA project must be imported into the workspace as follows:

1. Window -> Perspective -> Open Perspective -> Other... -> Choose Git -> right click on the Git repositories perspective -> Clone a Git Repository... -> copy `https://github.com/anonbnr/UML2RCA.git` into the URI field -> Next -> Enter GitHub credentials -> Next -> Select the proper directory to import the project -> Finish -> Enter GitHub credentials
2. Window -> Perspective -> Open Perspective -> Java -> right click on the Package Explorer perspective -> Import... -> Git -> Projects from Git -> Next -> Existing local repository -> UML2RCA -> Next -> Next -> Finish

2.3 Creating a new project using UML2RCA and RCA

Once Eclipse IDE and its required tools have been installed, and the RCA metamodel and UML2RCA projects have been imported into the workspace, one can create a modeling project that uses them in its code, as follows:

1. File -> New -> Other... -> Eclipse Modeling Framework -> Ecore Modeling Project, then follow the installation wizard.
2. right click on the created project -> Configure -> Convert to Maven Project -> fill the fields of the Maven project to be created, mainly GroupId, ArtifactId, Version, Packaging and optionally Name and Description -> Finish
3. open `pom.xml` and add the following at the end of the `<project></project>` root tag:

```
<!-- pom.xml of the modeling project -->
<project>
...
```

```

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org</groupId>
    <artifactId>rca</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>

  <dependency>
    <groupId>org</groupId>
    <artifactId>uml2rca</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
</dependencies>
...
</project>

```

2.3.1 Notes

1. If Eclipse got stuck upon adding the dependencies in the project's pom.xml, you can add the dependencies in the file outside Eclipse then **refresh the project** from within Eclipse.
2. Anytime an error arises from using an unrecognized element in the project, click **fix project setup...** in the proposed solutions, and choose the proposed project setup fix (*e.g. add X to required bundles*).

3 FCA (Formal Context Analysis)

3.1 Introduction

FCA is a data analysis and classification method used in many domains of computer sciences (*e.g. knowledge representation, knowledge engineering, information retrieval, supervised/unsupervised machine learning, ...*), based on the mathematical theories of order and lattices. It is mainly used to formally represent the notions of formal contexts and formal concepts.

3.2 Formal contexts

A **formal context** K is a triplet (O, A, I) , where O is a **set of objects**, A is a **set of boolean formal attributes**, and $I \subseteq O \times A$ is a **binary incidence**

relation, such that every couple (o, a) belonging to I indicates that the object o has the attribute a .









Drone	Gimbal	GPS	GLONASS	Avoidance	Headless	Altitude Hold	FT 10	FT ge 10	FT ge 20
Syma X4S Assault 					×		×		
Syma X8G 					×		×		
Parrot Bebop 		×				×		×	
DJI Ryze Tello 						×		×	
Hubsan X4 H502S 		×			×	×		×	
Aosenma CG035 GPS FPV 	×	×			×	×		×	
DJI Mavic Air 	×	×	×	×	×	×			×
Yuneec Typhoon H Pro 	×	×	×	×	×	×			×

Figure 1: “Formal Context Example”

3.3 Formal concepts

Given a formal context $K = (O, A, I)$, a **formal concept** C of K is a couple (E, I) , where $E \subseteq O$ is called the **extension** of the concept, and $I \subseteq A$ is called the **intension** of the concept. More specifically, C represents a maximum set of objects sharing a maximum set of attributes. Moreover, the set of all formal concepts associated to K is denoted by C_K .

3.4 Subsumption relation

Given a formal context $K = (O, A, I)$, **FCA** defines a **subsumption relation**, designating a **partial order relation**, denoted by \leq_S , over the set of all the formal concepts associated to it C_K . The order is based on the set inclusion of the concepts’ extensions and dualistically on the inverse set inclusion of the concepts’ intensions. It can also be interpreted as a formal concept specialization/generalization such that:

1. a concept is considered **more general** than another concept if it contains more objects in its extension, while sharing a reduced number of attributes.
2. a concept is considered **more specific** than another concept if it contains less objects in its extension, while sharing a larger number of attributes.

Concept_Drone_8	○	Concept identifier
Altitude Hold FT ge 10	○	Intent
DJI Ryze Tello Parrot Bebop Hubsan X4 H502S Aosenma CG035 GPS FPV	○	Extent





Drone	Gimbal	GPS	GLONASS	Avoidance	Headless	Altitude Hold	FT I 10	FT ge 10	FT ge 20
syma X4S Assault					×		×		
Syma X8G					×		×		
Parrot Bebop 		×				×		×	
DJI Ryze Tello 						×		×	
Hubsan X4 H502S 		×			×	×		×	
Aosenma CG035 GPS FPV 	×	×			×	×		×	
DJI Mavic Air	×	×	×	×	×	×			×
Yuneec Typhoon H Pro	×	×	×	×	×	×			×

Figure 2: “Formal Concept Example”

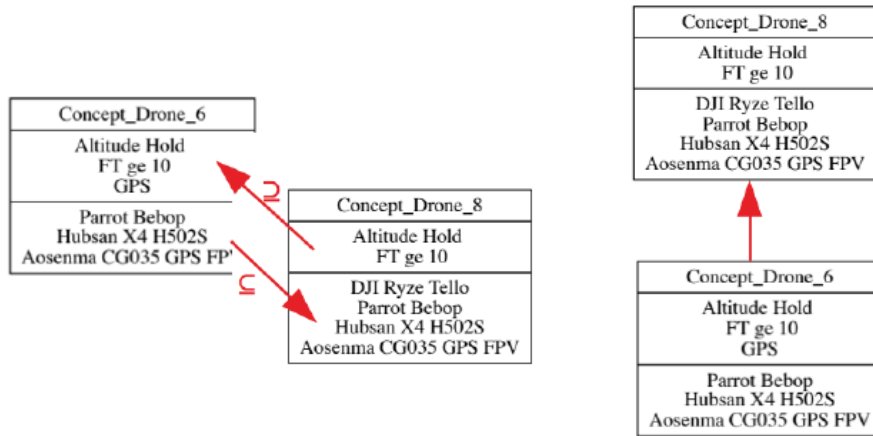


Figure 3: “Subsumption Relation Example”

3.5 Concept lattices

Given a formal context $K = (O, A, I)$, the subsumption relation \leq_S over C_K organizes the formal concepts in a **complete lattice** called a **concept lattice** or **Galois lattice**, denoted by (C_K, \leq_S) . It relies on a simplified attribute/object inheritance between the lattice's concepts.

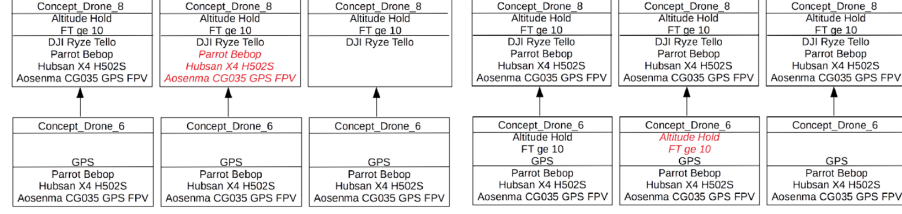


Figure 4: “Object and Attribute Inheritance Example”

3.6 Attribute/Object-introducing and neutral concepts

An **object-introducing concept** is a concept having at least one object in its **simplified extension**, whereas an **attribute-introducing concept** is one having at least one object in its **simplified extension**. On the other hand, A **neutral concept** is one that is neither an object-introducing concept nor an attribute-introducing concept.

Furthermore, given a formal context $K = (O, A, I)$, the set of all object-introducing concepts of K is denoted by OC_K , whereas its set of all attribute-introducing concepts is denoted by AC_K .

3.7 OC-posets, AC-posets and AOC-posets

Given that the number of concepts in a concept lattice for a formal context increases exponentially with an increase in the sizes of the provided input sets of objects and attributes, it could be useful to represent the concepts hierarchy using other hierarchical structures of representation. In particular, such structures are partially ordered sets (or **posets**), designating suborders that exclude neutral concepts from the complete concept lattice, and that don't necessarily constitute concept lattices themselves. Examples of such posets include:

1. **OC-posets (Object-introducing Concept partially ordered sets):** the suborder underlying the set of object-introducing concepts.
2. **AC-posets (Attribute-introducing Concept partially ordered sets):** the suborder underlying the set of attribute-introducing concepts.
3. **AC-posets (Attribute-Object-introducing Concept partially ordered sets):** the suborder underlying the set of attribute-introducing and object-introducing concepts.

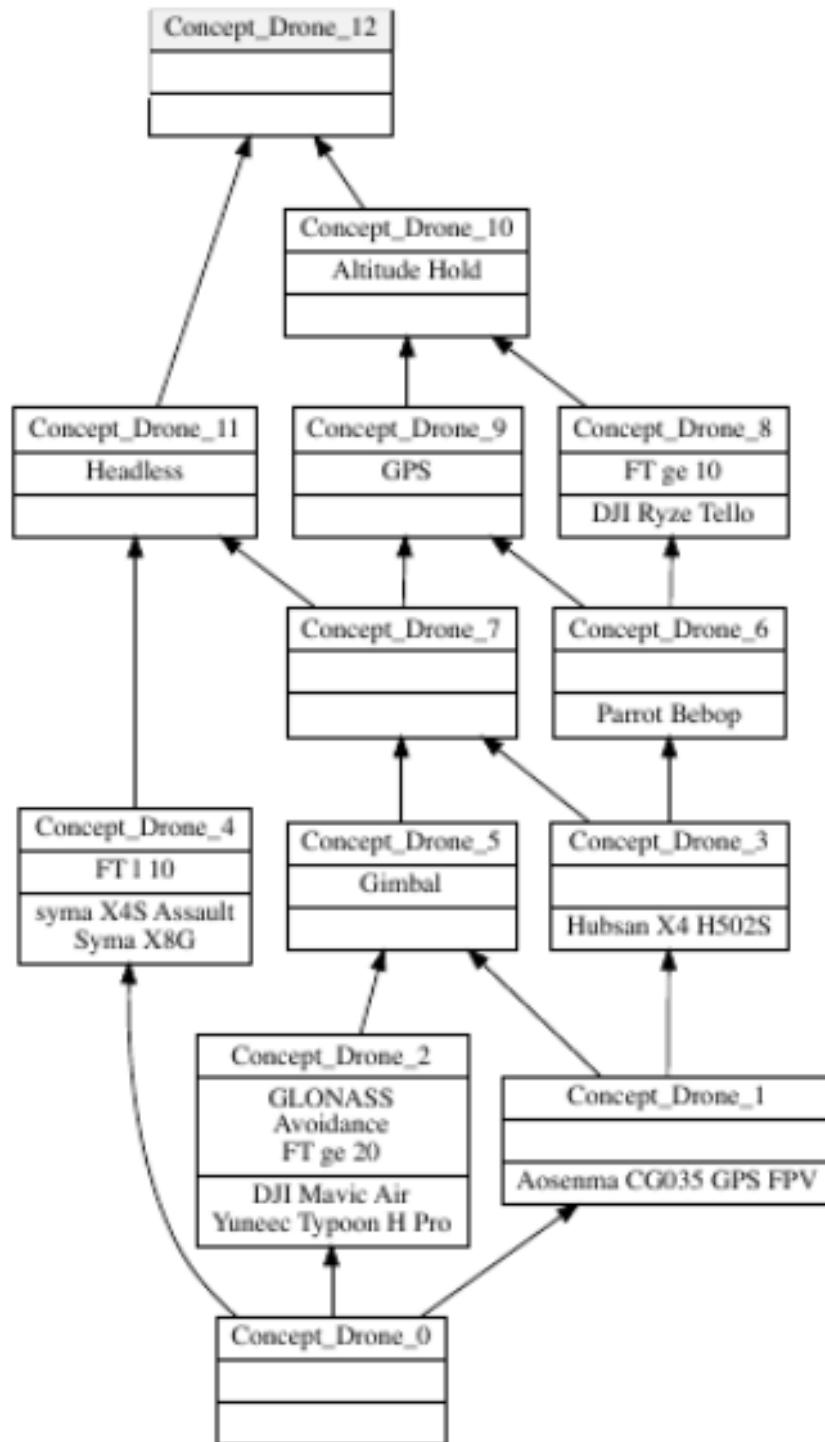


Figure 5: “concept lattice Example”

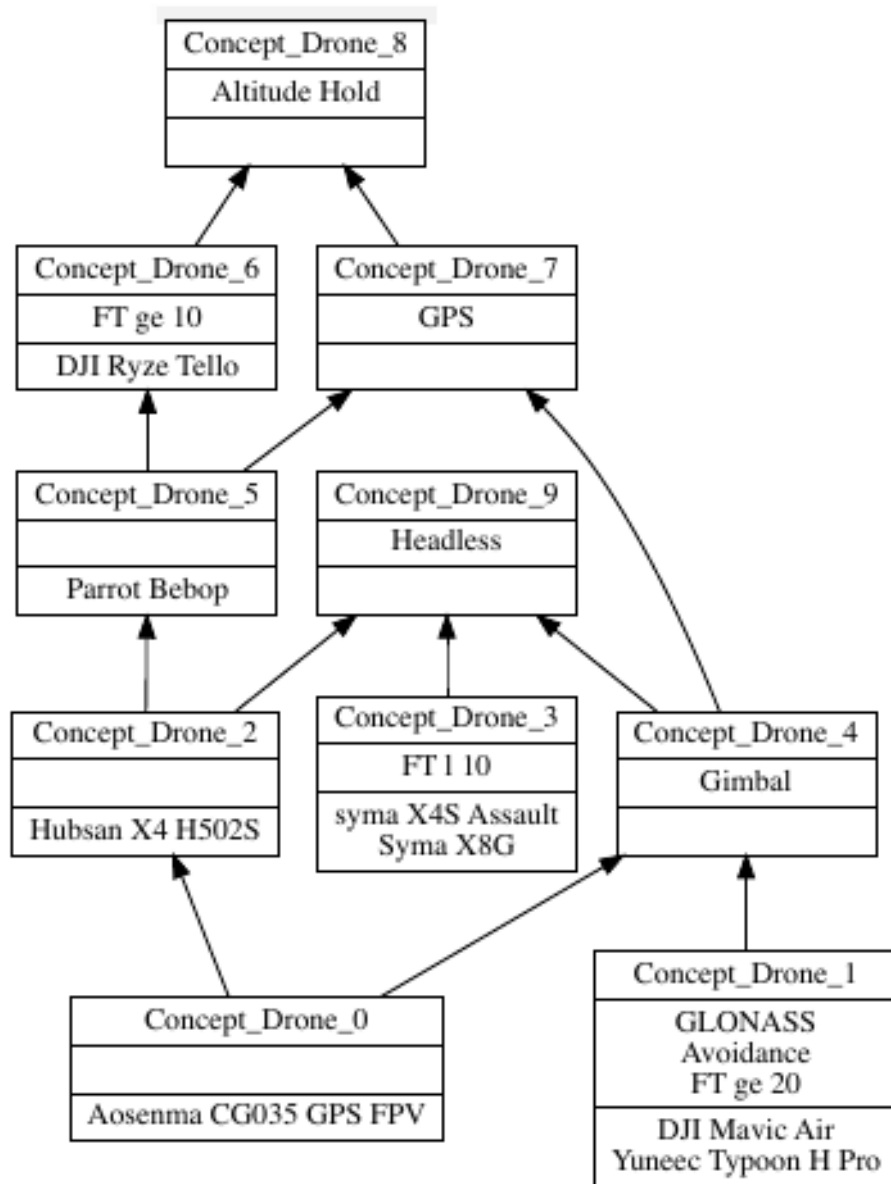


Figure 6: “AOC-poset Example”

4 RCA (Relational Context Analysis)

4.1 Introduction

RCA is an extension of **FCA** for relational datasets. It allows taking relationships between different categories of objects into consideration throughout the analysis, where each object category is designated by a **formal context** whose concepts are represented by a **concept lattice**, and each relationship is materialized through a **relational context**, designating directed binary links between two concept lattices. The related concept lattices represent the source and target formal contexts of the corresponding relational context.

Underlying RCA is an iterative process aiming to refine the relationships between concept lattices, representing source and target formal contexts of relational contexts, by scaling the relational contexts using **scaling operators**. As a consequence, the scaling operation generates **relational attributes** for each iteration that are added to source formal contexts, and allows to discover new concepts, until no new concept can be discovered. At the end of the process, a set of interconnected concept lattices is obtained, which can be represented in a hierarchy of ontological concepts that can be examined and analyzed using some knowledge representation formalism (*e.g. description logic*).

RCA can be very useful in software engineering, knowledge representation, artificial intelligence, ... Some applications include refactoring UML class and use case diagrams, extracting OO architectures, web service classification, construction and extraction of ontologies, ...

4.2 Relational contexts

A relational context R is a triplet (O_1, O_2, I) where O_1 is a set of objects originating from a formal context $K_1 = (O_1, A_1, I_1)$ called R 's **source context**, O_2 is a set of objects originating from a formal context $K_2 = (O_2, A_2, I_2)$ called R 's **target context**, and where $I \subseteq O_1 \times O_2$ is a **binary incidence relation**, such that every couple (o_1, o_2) belonging to I indicates that the object o_1 is related to the object o_2 .

4.2.1 Note

The source and target contexts of a relational context can designate the same formal context.

4.3 RCFs (Relational Context Families)

An RCF is a couple (K, R) , where K is a set of formal contexts, and R is a set of relational contexts relating formal contexts from K .

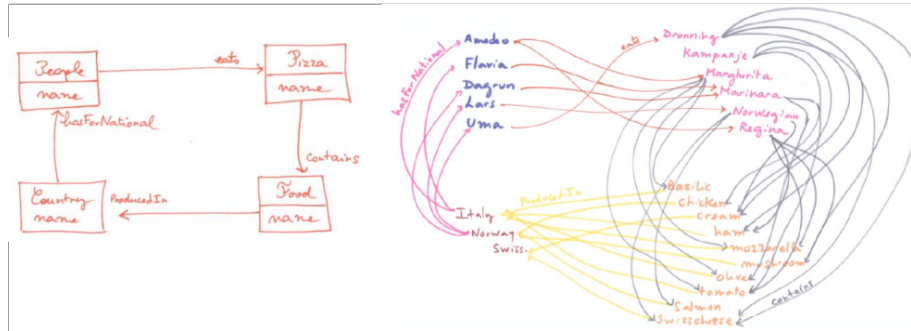


Figure 7: “Example of an RCF model”

K_{People}

	Amedeo	Flavia	Dagrun	Lars	Uma
Amedeo	x				
Flavia		x			
Dagrun			x		
Lars				x	
Uma					x

K_{Pizza}

	Dronning	Kampanje	Margherita	Marina	Norwegian	Regina
Dronning	x					
Kampanje		x				
Margherita			x			
Marina				x		
Norwegian					x	
Regina						x

K_{Food}

	basilic	chicken	cream	ham	mozzarella	mushroom	olive	tomato	salmon	swisscheese
basilic	x									
chicken		x								
cream			x							
ham				x						
mozzarella					x					
mushroom						x				
olive							x			
tomato								x		
salmon									x	
swisscheese										x

K_{Country}

	Italy	Norway	Switzerland
Italy	x		
Norway		x	
Switzerland			x

Figure 8: “Formal Contexts of the RCF example”

R_{eats}						
	Dronning	Kampanje	Margherita	Marina	Norwegian	Regina
Amedeo			×			×
Flavia				×		
Dagrun				×		×
Lars					×	
Uma	×				×	

$R_{\text{producedIn}}$			
	Italy	Norway	Switzerland
basilic	×		
chicken		×	
cream			×
ham	×		
mozzarella	×		
mushroom		×	
olive	×		
tomato	×		
salmon		×	
swisscheese			×

$R_{\text{HasNational}}$					
	Amedeo	Flavia	Dagrun	Lars	Uma
Italy	×	×			
Norway			×	×	×
Switzerland					

R_{contains}										
	basilic	chicken	cream	ham	mozzarella	mushroom	olive	tomato	salmon	swisscheese
Dronning			×	×		×				×
Kampanje		×	×							×
Margherita	×	×	×	×			×	×		
Marina							×	×		
Norwegian			×						×	
Regina				×	×	×		×		

Figure 9: “Relational Contexts of the RCF example”

4.4 Relational Scaling

Given an RCF (K, R) , we can refine the relationships between the concept lattices representing the concepts of its formal contexts, by scaling its relational contexts using **scaling operators**. For each relational context $R_i = (O_j, O_k, I_i) \in R$, we define a **scaled relational context** $R_i^* = (O_j, C_{K_k}, CI_i)$, where:

1. O_j is the set of objects of R_i ’s source formal context $K_i \in K$;
2. C_{K_k} is the set of concepts of the concept lattice built on the set of objects of R_i ’s target formal context $K_k \in K$;
3. $CI_i \subseteq O_j \times C_{K_k}$ is an **incidence binary relationship**, such that every couple $(o, c) \in CI_i \iff S(R_i(o), Extent(c))$ is true;
4. S is a **scaling operator**;
5. $R_i(o)$ is the set of objects in O_k , such that $\forall o' \text{ in } O_k, (o, o') \in I_i$.

4.4.1 Scaling operators & relational attributes

A **scaling operator** is an operator applied on a relational context to scale, that generates relational attributes in its source formal context, allowing to discover new formal concepts therein iteratively, and therefore refine the relationship between the concept lattices of the source and target formal contexts of the scaled relational context, until no new concepts can be discovered. In particular, a scaling operator is a predicate having an object, a relational context to scale, and a concept as input, and indicating whether the object and the concept are related by the scaled-up binary incidence relationship of the scaled relational context.

Relational attributes are binary attributes generated by scaling a relational

context using a scaling operator, and having the generic syntax: $S \text{ [arg] } r : c$, where:

1. S is a scaling operator;
2. **arg** is an optional argument specific to S ;
3. r is the relational context scaled by S ;
4. c is a concept in the set of concepts of the concept lattice built on the set of objects of r 's source target formal context.

4.4.2 Types of scaling operators

Operator	Relational Attribute Syntax	Predicate Condition
Universal (wide)	$\forall r : c$	$r(o) \subseteq \text{Extension}(c)$
Universal (strict)	$\forall \exists r : c$	$r(o) \subseteq \text{Extension}(c)$ and $r(o) \neq \emptyset$
Existential	$\exists r : c$	$r(o) \cap \text{Extension}(c) \neq \emptyset$
Includes	$\supseteq r : c$	$r(o) \supseteq \text{Extension}(c)$
Cardinality restriction	$\geq nr : c$	$r(o) \subseteq \text{Extension}(c)$ and $ r(o) \geq n$
...

4.4.3 Properties of relational scaling

1. the homogeneity of concept descriptions is preserved, since all attributes (formal and relational attributes) are considered binary;
2. the standard algorithms for building concept lattices can be directly reused throughout the iterative RCA process.

4.4.4 Example of relational scaling: The existential scaling operator

Given an RCF (K, R) , for each relational context $R_i = (O_j, O_k, I_i) \in R$, we define a **scaled relational context** $R_i^* = (O_j, C_{K_k}, CI_i)$, using the existential scaling operator, as follows:

1. $CI_i \subseteq O_j \times C_{K_k}$ is an **incidence binary relationship**, such that every couple $(o, c) \in CI_i \iff S(R_i(o), \text{Extent}(c))$ is true;
2. $S(R_i(o), \text{Extent}(c))$ is true $\iff \exists x \in R(o), x \in \text{Extent}(c)$

The first step consists of using the RCA's formal contexts to create their initial concept lattices. Afterwards, for each iteration i , we apply the existential scaling operator to scale the RCA's relational contexts, in order to generate the relational attributes, discover new concepts to refine the concept lattices of the iteration $i - 1$, and obtain the new refined concept lattices of the iteration i . Once no new concepts can be discovered, the process terminates and we are left with a set of interconnected and refined concept lattices upon which we can

apply knowledge exploration algorithms (*among others*).

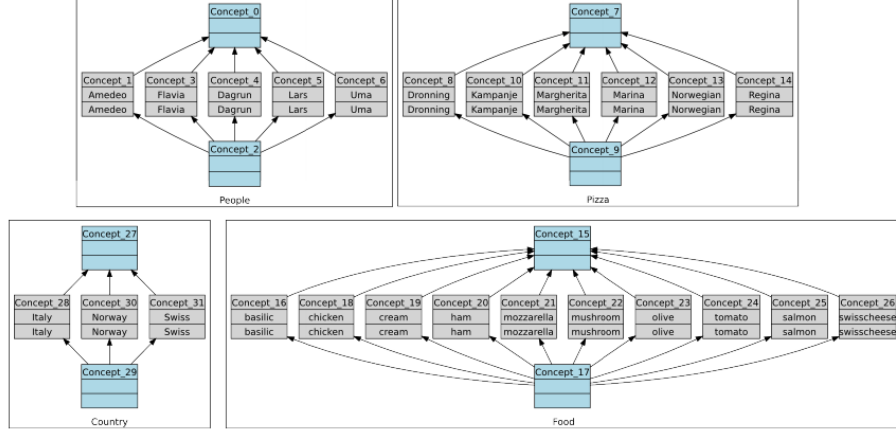


Figure 10: “Initial concept lattices of the RCF example”

4.5 RCA iterative algorithm

Algorithm 1: Multi-FCA

Input: An RCF (K, R)

Result: A family of 1 to n interconnected concept lattices

```

1 begin
2    $p \leftarrow 0$ ;
3   halt  $\leftarrow$  false;
4   for  $i \leftarrow 1$  to  $n$  do
5      $\mathbb{L}^0[i] \leftarrow \text{BUILD-LATTICE}(\mathbb{K}_i^0)$ ;
6   while not halt do
7      $p++$ ;
8     for  $i \leftarrow 1$  to  $n$  do
9        $\mathbb{K}_i^p \leftarrow \text{EXTEND-REL}(\mathbb{K}_i^{p-1}, \mathbb{L}^{p-1})$ ;
10       $\mathbb{L}^p[i] \leftarrow \text{UPDATE-LATTICE}(\mathbb{K}_i^p, \mathbb{L}^{p-1}[i])$ ;
11    halt  $\leftarrow \bigwedge_{i=1}^n \text{ISOMORPHIC}(\mathbb{L}^p[i], \mathbb{L}^{p-1}[i])$ ;

```

5 The RCA Metamodel

Based on the brief aforementioned introduction to RCA, the following meta-model has been crafted to describe its entities and associations.

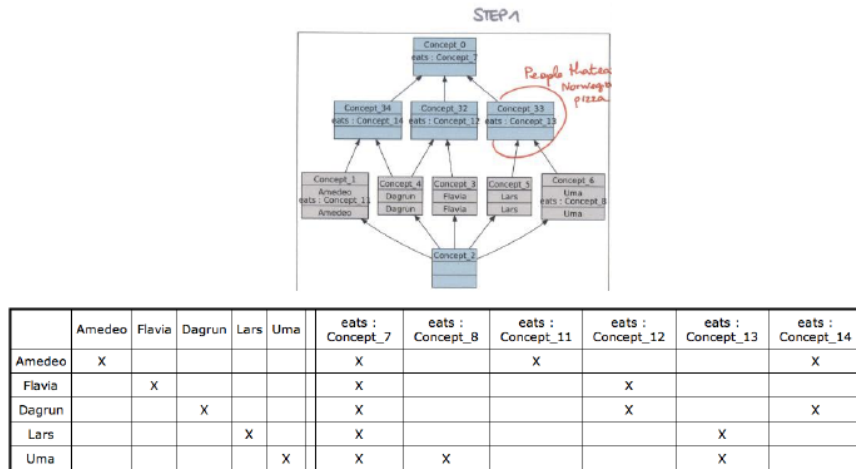


Figure 11: “Concept lattice obtained after the first iteration of scaling the relational context ‘eats’”

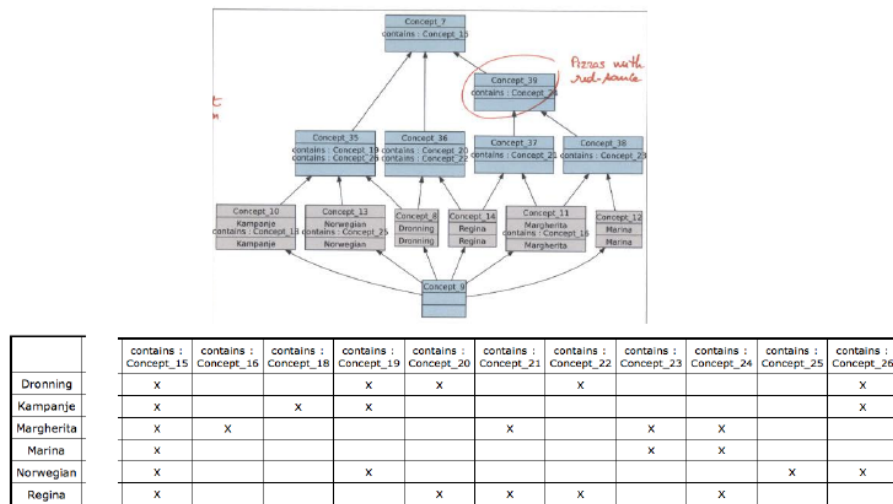
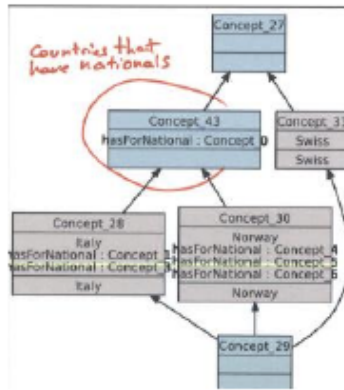


Figure 12: “Concept lattice obtained after the first iteration of scaling the relational context ‘contains’”



	producedIn : Concept_27	producedIn : Concept_28	producedIn : Concept_30	producedIn : Concept_31
basilic	X	X		
chicken	X		X	
cream	X			X
ham	X	X		
mozzarella	X	X		
mushroom	X		X	
olive	X	X		
tomato	X	X		
salmon	X		X	
swisscheese	X			X

Figure 13: “Concept lattice obtained after the first iteration of scaling the relational context ‘producedIn’ ”

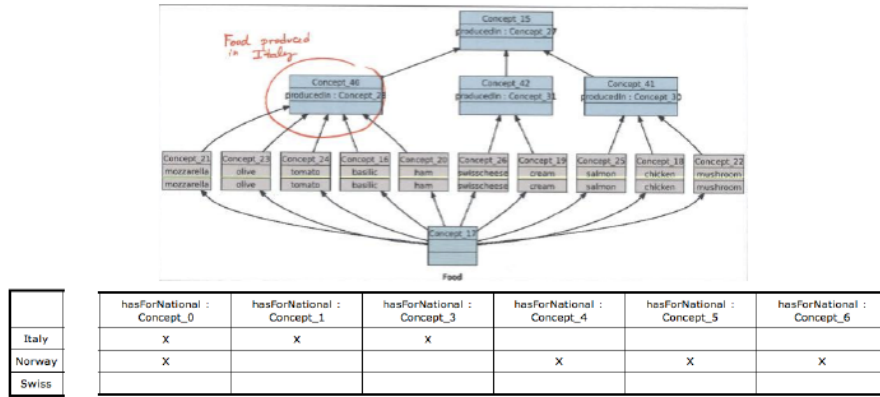


Figure 14: “Concept lattice obtained after the first iteration of scaling the relational context ‘hasForNational’ ”

5.1 The Core viewpoint

TODO

5.1.1 Description

TODO

5.2 The Formal Context viewpoint

TODO

5.2.1 Description

TODO

5.3 The Relation Context viewpoint

TODO

5.3.1 Description

TODO

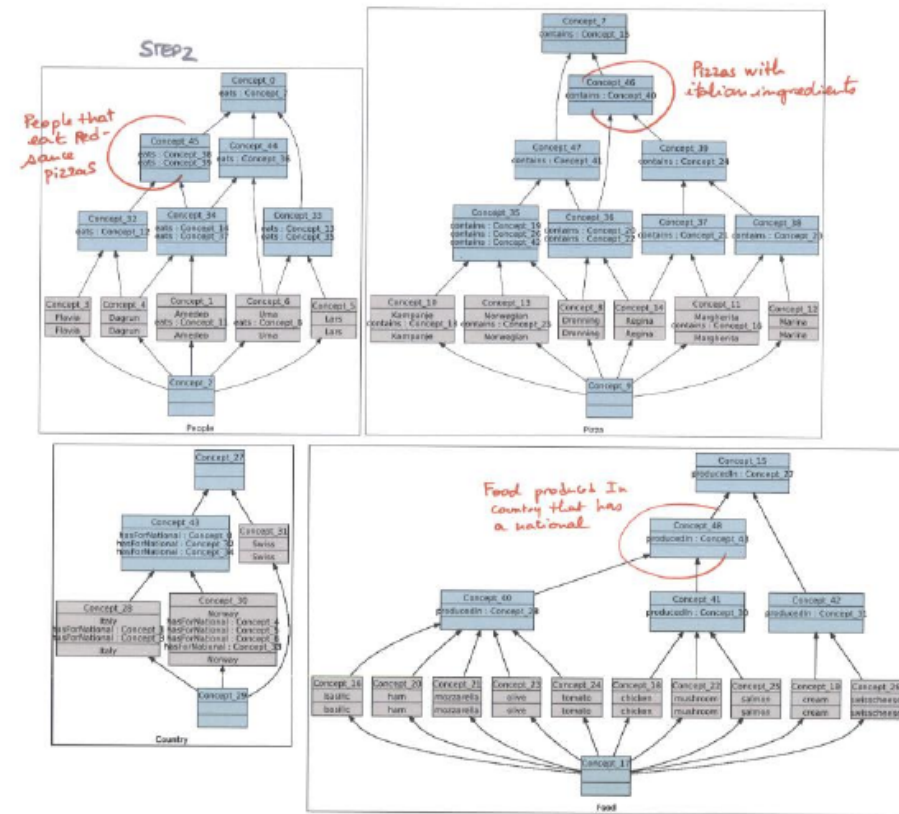


Figure 15: “Concept lattices obtained after the second iteration of scaling all relational contexts”

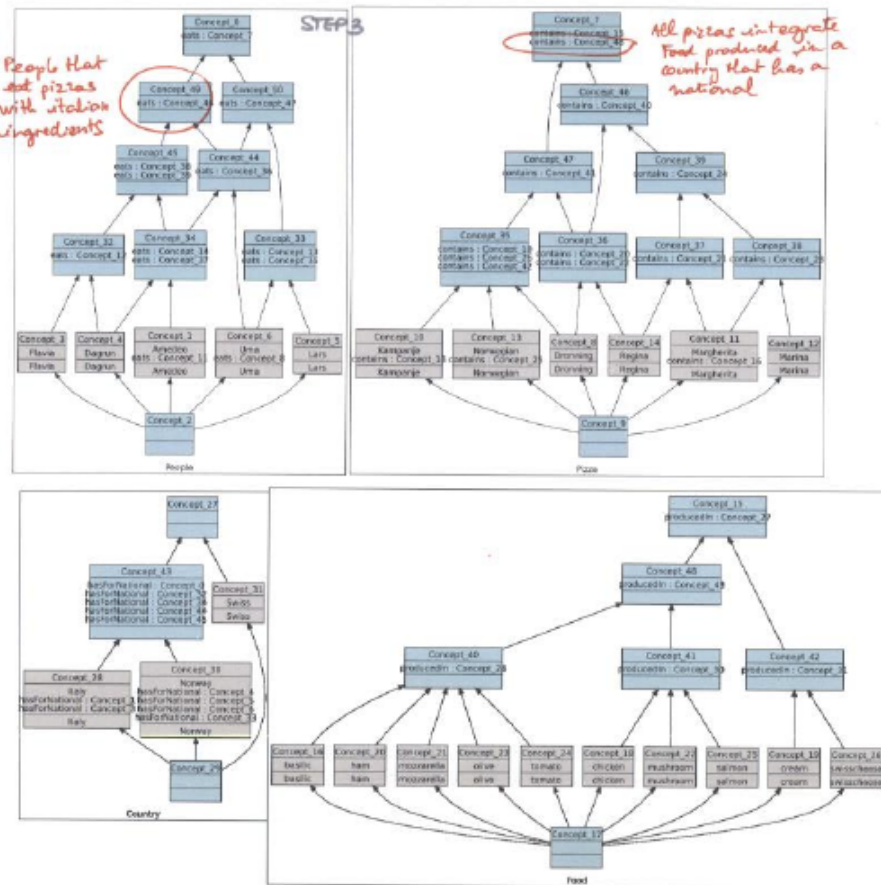


Figure 16: "Concept lattices obtained after the third iteration of scaling all relational contexts"

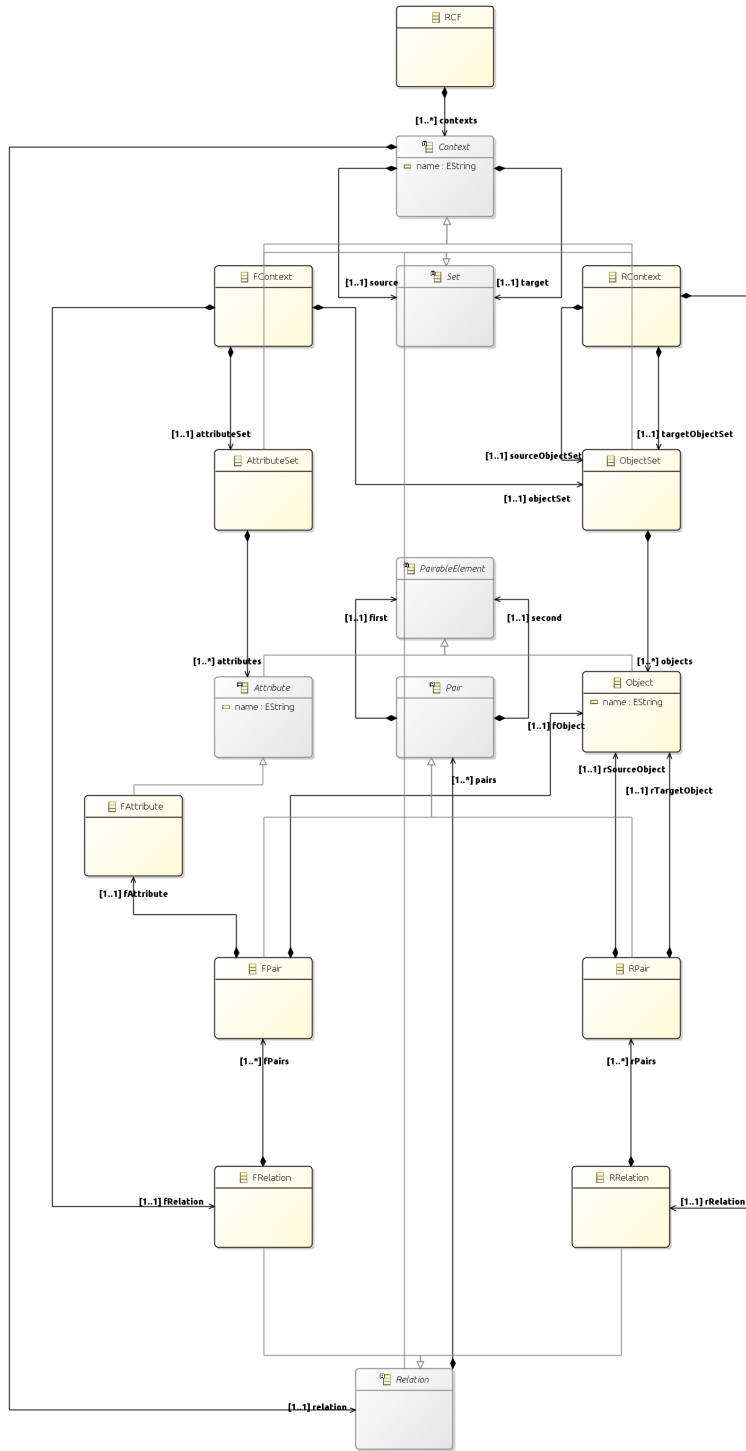


Figure 17: The Global viewpoint of the RCA Metamodel

5.4 The Global viewpoint

6 The Generic Metamodel Transformation Architecture

6.1 Introduction

TODO

6.2 Transformations, Transformation strategies, and Transformers

TODO

6.3 Adaptations, Adaptation strategies, and Adapters

TODO

6.4 Conversions, Conversion strategies, and Converters

TODO

6.5 Model Management

TODO

6.6 Conflict Management

TODO

7 The UML-to-RCA use-case

7.1 Introduction

TODO

7.2 Model Management

TODO

7.2.1 Ecore Model Manager

TODO

7.2.2 Ecore Model State

TODO

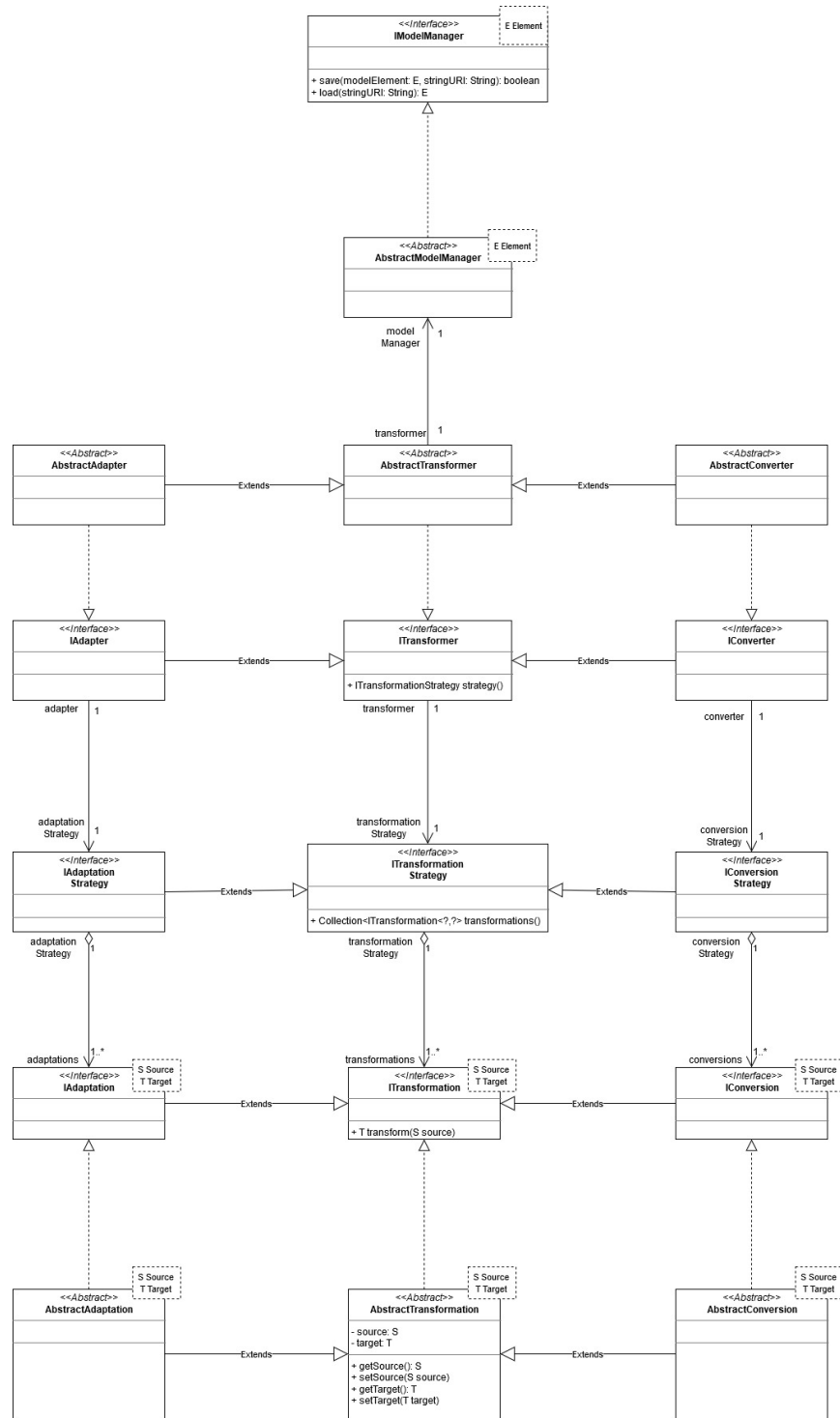


Figure 18: The Generic Metamodel Transformation Architecture

7.3 Adaptations (X-UML to R-UML)

TODO

7.3.1 Associations & Dependencies

TODO

7.3.1.1 Aggregations *TODO*

7.3.1.2 Compositions *TODO*

7.3.1.3 Dependencies *TODO*

7.3.1.4 Association classes *TODO*

7.3.1.5 Bidirectional associations *TODO*

7.3.1.6 N-ary associations *TODO*

7.3.1.6.1 The Materialization solution *TODO*

7.3.1.6.2 The Forgotten solution *TODO*

7.3.1.7 Associations with abstract members *TODO*

7.3.1.8 Dependencies with abstract members *TODO*

7.3.2 Generalizations

TODO

7.3.2.1 Simple Generalizations *TODO*

7.3.2.2 Multiple Generalizations *TODO*

7.3.2.3 Visitors *TODO*

7.3.2.4 Conflict Management *TODO*

7.4 Conversions (R-UML to unpopulated RCFs)

TODO

7.4.1 Associations to Relational Contexts

TODO

7.4.2 Concrete Classes to Formal Contexts

TODO

7.4.3 Boolean Attributes to Formal Attributes

TODO

7.5 Instance migrations (populating the RCFs)

TODO