



Pós-Graduação em Ciência da Computação

Estela Domingues Nunes da Silva

**Structural Validation of Enhanced Entity-Relationship Models using Description
Logic Reasoners**



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
<http://cin.ufpe.br/~posgraduacao>

Recife
2019

Estela Domingues Nunes da Silva

**Structural Validation of Enhanced Entity-Relationship Models using Description
Logic Reasoners**

Work presented to the Graduate Program in
Computer Science of the Federal University of
Pernambuco, as a partial requirement to obtain
the title of Master of Science in Computer Sci-
ence.

Area: Databases

Supervisor: Robson do Nascimento Fidalgo

Co-supervisor: Frederico Luiz Gonçalves de
Freitas.

Recife
2019

Catálogo na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586s Silva, Estela Domingues Nunes da
*Structural validation of enhanced entity-relationship models using description
logic reasoners* / Estela Domingues Nunes da Silva. – 2019.
98 f.: il., fig., tab.

Orientador: Robson do Nascimento Fidalgo.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn,
Ciência da Computação, Recife, 2019.
Inclui referências e apêndices.

1. Banco de dados. 2. Validação de modelos conceituais. I. Fidalgo, Robson
do Nascimento (orientador). II. Título.

025.04

CDD (23. ed.)

UFPE - CCEN 2020 - 147

Estela Domingues Nunes da Silva

**“Structural validation of Enhanced Entity-Relationship Models Using
Description Logic Reasoners”**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 3 de setembro de 2019.

BANCA EXAMINADORA

Prof. Dr. Kiev Santos da Gama
Centro de Informática/UFPE

Profa. Dra. Natasha Correia Queiroz Lino
Centro de Informática / UFPB

Prof. Dr. Robson do Nascimento Fidalgo
Centro de Informática / UFPE
(Orientador)

ACKNOWLEDGEMENTS

First, I would like to praise and thank God, the Almighty, for His goodness, guidance, and blessings throughout this journey. Furthermore, I would like to thank my supervisor, Prof. Robson Fidalgo, for his invaluable support, encouragement, and recommendations, without which it would not be possible to complete this work. Also, I would like to acknowledge my co-supervisor, Prof. Frederico Freitas, for his availability, and essential guidance for conducting this research. Finally, a special thanks to my mother, father, sister, and husband for their kind support, patience, and love during this intense period.

ABSTRACT

The Enhanced Entity-Relationship (EER) language is widely used in the creation of conceptual database models. The validation of these models is critical as validity errors can be passed to the next phases of the project and negatively influence the outcome. In large and complex models, validation becomes a difficult task because the interaction between the elements used can produce inconsistencies and unintended implicit consequences. Hence, it is essential to offer automatic assistance. Description Logics (DLs) are a set of languages used for knowledge representation. They admit decidable and automated reasoning tasks, such as the identification of implicit logical consequences. Because of those characteristics, DLs have been considered a promising alternative to represent and reason on conceptual models. This work aims to support the validation of conceptual database models by identifying syntactic and semantic inconsistencies in EER models using DL reasoners. To the best of our knowledge, few work use Description Logics to represent and reason on EER models. Also, these work do not cover aspects such as the interaction between model constraints and the related structural consequences. Our work stands out for taking into account the consequences of constraints such as cardinality, participation, relationship type degree, inheritance, cyclic paths, and valid attribute types, as well as the consequences of the interactions between these constraints on the same model. With the support of Protégé, we built a Knowledge Base(KB) in OWL DL by formalizing the EER syntax. Next, we added the semantic validity rules related to the constraints mentioned. Although we tried to represent most of the rules by using axioms, we also made use of Semantic Web Rule Language (SWRL) rules in cases in which DL expressivity was not sufficient. Finally, we manually converted the KB to ALCROIQ language. As proof of concept, we successfully validated case studies by using DL reasoners.

Keywords: Conceptual Model Validation. Database Modeling. Reasoning.

RESUMO

A linguagem Entidade-Relacionamento Estendido (EER) é amplamente utilizada na criação de modelos conceituais de banco de dados. A validação desses modelos é crítica, pois os erros de validação podem ser passados para as próximas fases do projeto e influenciar negativamente o resultado final. Em modelos grandes e complexos, a validação torna-se uma tarefa difícil porque a interação entre os elementos utilizados pode produzir inconsistências e consequências implícitas não intencionais. Por isso, é essencial oferecer assistência automática. A Lógica de Descrição (DL) é um conjunto de linguagens utilizadas para representação de conhecimento. Ela admite tarefas de raciocínio decidíveis e automatizadas, como a identificação de consequências lógicas implícitas. Por causa dessas características, a DL tem sido considerada uma alternativa promissora para representar e raciocinar em modelos conceituais. Este trabalho tem como objetivo apoiar a validação de modelos conceituais de banco de dados, identificando inconsistências de sintaxe e semântica em modelos EER utilizando racionalizadores de DL. De acordo com as pesquisas realizadas, poucos trabalhos usam Lógica de Descrição para representar e raciocinar sobre modelos EER. Além disso, esses trabalhos não abordam aspectos como a interação entre as restrições da linguagem em um mesmo modelo e suas consequências estruturais. Nosso trabalho se destaca por levar em consideração as consequências de restrições como cardinalidade, participação, grau do tipo de relacionamento, herança, caminhos cíclicos e tipos de atributos válidos, bem como as consequências das interações entre essas restrições. Com o apoio da ferramenta Protégé, construímos uma Base de Conhecimento (BC) em OWL DL, formalizando a sintaxe da linguagem EER. Em seguida, adicionamos as regras de validação semântica relacionadas às restrições mencionadas. Embora a maioria das regras tenham sido representadas por meio de axiomas, também utilizamos regras SWRL (Semantic Web Rule Language) em casos nos quais a expressividade de DL não era adequada. Ao final, convertimos manualmente a BC para a linguagem ALCROIQ. Como prova de conceito, validamos com sucesso estudos de caso utilizando raciocinadores DL.

Palavras-chaves: Validação de Modelos Conceituais. Modelagem de Banco de Dados. Raciocínio.

LIST OF FIGURES

Figure 1 – EERMM Notation	16
Figure 2 – Employee entity type and its attributes	17
Figure 3 – Relationship example	18
Figure 4 – Model example with roles	18
Figure 5 – Model example with cardinality and participation constraints	19
Figure 6 – Weak entity type example	19
Figure 7 – Weak entity type example with discriminator attribute	20
Figure 8 – Inheritance notations	20
Figure 9 – Inheritance notations	21
Figure 10 – SWRL Rule Example	25
Figure 11 – Semantically invalid model	26
Figure 12 – Invalid model according to R01	31
Figure 13 – Invalid model according to R02	31
Figure 14 – Invalid model according to R03	31
Figure 15 – Invalid model according to R04	32
Figure 16 – Invalid model according to R05	34
Figure 17 – Invalid model according to R06	34
Figure 18 – Invalid model according to R07	35
Figure 19 – Invalid model according to R08	35
Figure 20 – Invalid model according to R09	35
Figure 21 – Invalid model according to R10	36
Figure 22 – Invalid model according to R11	36
Figure 23 – Invalid model according to R12	37
Figure 24 – Implicit binary relationships in ternary relationships	37
Figure 25 – Ternary relationship example	38
Figure 26 – Invalid model according to R13	38
Figure 27 – Binary equivalent model	38
Figure 28 – Invalid model according to R14	39
Figure 29 – Invalid model according to R15	40
Figure 30 – Invalid model according to R16	40
Figure 31 – Invalid model according to R16	40
Figure 32 – Invalid models according to R17	41
Figure 33 – Invalid model according to R18	41
Figure 34 – Invalid model according to R19	42
Figure 35 – Invalid model according to R20	42
Figure 36 – Invalid model according to R21	43

Figure 37 – Invalid model according to R22	43
Figure 38 – Invalid model according to R23	44
Figure 39 – Invalid model according to R24	44
Figure 40 – Invalid model according to R25	45
Figure 41 – Invalid model according to R26	45
Figure 42 – Invalid model according to R27	45
Figure 43 – EERMM Metamodel	51
Figure 44 – EERMM concepts	52
Figure 45 – EERMM concepts	52
Figure 46 – SWRL Rule: Case 2	56
Figure 47 – SWRL Rule: Case 9	56
Figure 48 – SWRL Rule: Case 10	56
Figure 49 – SWRL Rule: isSmallerThan transitivity	56
Figure 50 – SWRL Rule: Case 1	57
Figure 51 – SWRL Rule: isEqualsto transitivity	57
Figure 52 – Weak and Strong Entities	58
Figure 53 – SWRL Rule: dependsOn	59
Figure 54 – SWRL Rule: dependsOn transitivity	59
Figure 55 – R11 Example	60
Figure 56 – SWRL Rule: linkedRelationships	60
Figure 57 – SWRL Rule: smallerCardinality	61
Figure 58 – R12 Example	61
Figure 59 – SWRL Rule: differentParticipation	62
Figure 60 – SWRL Rule: isSpecializationOf transitivity	63
Figure 61 – Protégé individuals tab	73
Figure 62 – Pellet reasoning result (Protégé)	74
Figure 63 – Hermit reasoning result (Protégé)	74
Figure 64 – Valid model according to R01	75
Figure 65 – Invalid model according to R01	75
Figure 66 – Pellet reasoning result (Protégé)	76
Figure 67 – Hermit reasoning result (Protégé)	76
Figure 68 – Inconsistent ontology explanation	77
Figure 69 – Pellet reasoning result (27 validation errors)	77
Figure 70 – Hermit reasoning result (27 validation errors)	78

LIST OF TABLES

Table 1 – \mathcal{AL} language constructors.	23
Table 2 – $\mathcal{ALCROIQ}$ language constructors.	24
Table 3 – EER structural validity rules catalog	27
Table 4 – Structural restrictions of a binary relationship.	33
Table 5 – EERMM Partial RBox	48
Table 6 – EERMM Partial TBox	49
Table 7 – <i>relation</i> and <i>iRelation</i> hierarchies	53
Table 8 – Summary of strategies: unary relationship types	54
Table 9 – Structural restrictions of a binary relationship.	55
Table 10 – Summary of strategies: binary relationship types	59
Table 11 – Summary of strategies: ternary relationship types	62
Table 12 – Summary of strategies: inheritance	63
Table 13 – Summary of strategies: attribute type	64
Table 14 – EERMM RBox	65
Table 15 – EERMM TBox	66
Table 16 – R01 valid model (ABox fragment in $\mathcal{ALCROIQ}$)	74
Table 17 – R01 invalid model (ABox fragment in $\mathcal{ALCROIQ}$)	75
Table 18 – Survey results summary	81
Table 19 – Invalid examples	90
Table 20 – Valid examples	95

CONTENTS

1	INTRODUCTION	12
1.1	CONTEXT	12
1.2	MOTIVATION	12
1.3	GOALS	13
1.4	METHODOLOGY	14
1.5	CONTRIBUTION	14
1.6	STRUCTURE	15
2	BACKGROUND	16
2.1	INTRODUCTION	16
2.2	ENHANCED ENTITY-RELATIONSHIP MODEL	16
2.3	EXISTENCE DEPENDENCY GRAPHS	21
2.4	DESCRIPTION LOGICS	22
2.4.1	<i>AL</i> Language	22
2.4.2	<i>ALCROIQ</i> Language	24
2.5	OWL DL AND THE SEMANTIC WEB RULE LANGUAGE	24
3	EER SEMANTIC RULES	26
3.1	INTRODUCTION	26
3.1.1	Validity Rules Catalog	26
3.1.2	Unary Relationship Type	30
3.1.3	Binary Relationship Type	32
3.1.4	Ternary Relationship Type	36
3.1.5	Inheritance	38
3.1.6	Attribute	42
3.2	FINAL REMARKS	46
4	EER KNOWLEDGE BASE	47
4.1	INTRODUCTION	47
4.2	ABSTRACT SYNTAX FORMALIZATION	47
4.3	SEMANTIC VALIDITY RULES FORMALIZATION	53
4.4	PROOF OF CONCEPT	72
5	RELATED WORK	79
5.1	INTRODUCTION	79
5.2	STATIC MODELS VALIDATION IN THE LITERATURE	79
5.3	SURVEY	81

5.4	FINAL REMARKS	82
6	CONCLUSION	83
6.1	INTRODUCTION	83
6.2	FINAL REMARKS	83
6.3	FUTURE WORK	84
	REFERENCES	86
	APPENDIX A – LIST OF INVALID EXAMPLES	90
	APPENDIX B – LIST OF VALID EXAMPLES	95

1 INTRODUCTION

This chapter presents the context, motivation, goals, methodology, and contributions of our work. It also describes the structure of the remaining chapters.

1.1 CONTEXT

A database design project usually consists of three phases: conceptual design, logical design, and physical design ((ELMASRI; NAVATHE, 2010), (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)). In the conceptual phase, the designers transform business requirements into conceptual models, focusing on the representation of business-related concepts and their relationships. In the logical phase, the designers map the conceptual model from the previous phase to an implementation data model, which varies according to the approach taken by the database technology (e.g., the relational data model). Finally, in the physical phase, the designers convert the logical model into a physical one consisting of defining how data will be stored and organized in files, which depends on the database technology chosen.

In this work, we concentrate on the conceptual design phase. In order to create conceptual models, designers make use of modeling languages. One of the most commonly used languages for conceptual database modeling is the Enhanced Entity-Relationship (EER) (ELMASRI; NAVATHE, 2010). It is an extension of the Entity-Relationship (ER) model proposed by Chen (1976), adding the concepts of associative entity, inheritance, and category, which are conceptual data modeling constructors also developed in other areas such as Knowledge Representation and Software Engineering (ELMASRI; NAVATHE, 2010). Due to its relevance and expressiveness, there are several research papers related to the EER language ((ZHANG; MA; CHENG, 2016), (FIDALGO et al., 2013), (VAGNER, 2018), (KOLP; ZIMANYI, 2000)), as well as many database textbooks that present a chapter about it ((ELMASRI; NAVATHE, 2010), (CONNOLLY; BEGG, 2010), (SINGH, 2009)).

1.2 MOTIVATION

As mentioned earlier, the conceptual model created in the first phase of a database project is used as a basis for the other phases. Therefore, validating conceptual models is crucial as validation errors can be passed to the next phases and negatively influence the project outcome. Moreover, the sooner those errors are identified, the lower the costs to fix them. According to Westland (2002), “uncorrected errors become exponentially more costly with each phase in which they are unresolved”.

As also stated before, designers use modeling languages to create conceptual models. A modeling language comprises three elements: abstract syntax, concrete syntax, and

semantics (BRAMBILLA; CABOT; WIMMER, 2012). The abstract syntax (or metamodel) describes the elements involved in the language, as well as their possible associations. The EER abstract syntax, for example, comprises elements such as entity type, relationship type, and attribute. The concrete syntax refers to the notation of the language, i.e., the ways that we represent the concepts and associations. In the EER, for example, we use a rectangle to represent an entity type. Finally, the semantics is related to "the meaning of the elements defined in the language and the meaning of the different ways of combining them" (BRAMBILLA; CABOT; WIMMER, 2012). The semantics of EER comprises, for example, the meaning of the entity type concept, and the meaning of the cardinality constraints in a relationship type, as we will discuss in chapter 2. Therefore, when checking the model validity, we must take into account the syntactic and semantic aspects, as well as the notation of the modeling language used.

In small projects, the designer can quickly identify and solve most of these syntactic and semantic problems. However, in large and complex ones, automatic support becomes necessary.

Description Logics (DLs) are a set of languages used for knowledge representation, allowing to represent concepts, roles, and individuals by creating Knowledge Bases (KBs). It admits decidable and automated reasoning tasks using specific tools called reasoners. By using DL reasoners, it is possible to verify automatically whether a KB is consistent, to establish hierarchies between concepts (subsumption), and to check concept equivalence. It is also possible to identify implicit logical consequences that would be hard to identify manually. Because of those characteristics, DLs have been considered a promising alternative to represent and reason on conceptual models ((GONZALEZ; CABOT, 2014), (ZHANG; MA; CHENG, 2016)).

To the best of our knowledge, few research studies use description logics to represent and reason in EER models. Also, those studies do not cover aspects as the semantics related to the combinations of constraints and model elements. Hence, our work stands out for considering the consequences of constraints such as cardinality, participation, relationship type degree, inheritance, cyclic paths, and valid attribute types, as well as the consequences of the interactions between those restrictions in the same model.

1.3 GOALS

This work aims to support the automatic validation of conceptual database models by identifying syntactic and semantic inconsistencies in EER models using DL reasoners. The automatic validation intends to prevent problems in the conceptual design phase from spreading to the other phases, avoiding increased project costs as well as unsatisfactory results. Based on that, our specific objectives are:

- To create a Knowledge Base that describes the EER syntax and its semantic rules,

focusing on the valid ways of combining elements;

- To verify test cases using automatic reasoners.

1.4 METHODOLOGY

In order to achieve our goals, we first identified primary studies related to the validation of conceptual models using DL, aiming to perceive their main approaches and the coverage of their solutions. We found that few studies focus on the validation of EER models. Also, we found that while some of them centered on the computational complexity aspects of model validation ((ARTALE et al., 2007a), (ARTALE et al., 2007b)), others missed semantic aspects such as the consequences of the interaction between cardinality, participation, relationship type degree, inheritance, cyclic paths, and valid attribute types (ZHANG; MA; CHENG, 2016).

Based on that, we created an EER KB by performing the following steps:

- We formalized an EER language syntax by creating a KB in OWL DL using Protégé (MUSEN, 2015). Since the EER language does not have a standard abstract syntax, we consider the EERMM one, which covers all EER constructs (FIDALGO et al., 2013);
- Next, we added the semantic validity rules organized by Nascimento (2015) and based on the work of Dullea, Song and Lamprou (2003) and Calvanese and Lenzerini (1994). Those rules are related to cardinality, participation, relationship type degree, inheritance, cyclic paths, valid attribute types, and these constraints interactions within the same model. Although we tried to represent most of the rules by using axioms, we also made use of Semantic Web Rule Language (SWRL) rules in the cases involving cyclic paths, since DL expressivity was not sufficient;
- After that, we created individuals based on a test case set composed of invalid and valid models (annex A and B, respectively) and used DL reasoners to identify the invalid constructs;
- Finally, we manually converted the KB to *ALCROIQ*.

1.5 CONTRIBUTION

The main contributions of our work are:

- The formalization of the EER language through DL, which allows validating the structure of EER models with the use of automatic reasoners;
- The possibility of reusing the Knowledge Base if new validation rules need to be added or in case of a language extension;

- The formalization of the EERMM, which contributes to a possible standardization of the EER abstract syntax.

1.6 STRUCTURE

We present the results of our work in the next chapters. In chapter 2 we briefly recall the basic concepts related to the Enhanced Entity-Relationship (EER) model, the EERMM metamodel, the Existence Dependency Graphs, Description Logics, OWL DL, and the Semantic Web Rule Language. In chapter 3, we present the EER structural validity rules mentioned. In chapter 4, we describe the EER Knowledge Base (KB) created in order to validate EER models, and the procedures conducted to verify our KB. In chapter 5, we present primary studies related to the validation of EER models using DL, as well as our main contributions. Finally, in chapter 6, we present our final remarks and future work.

2 BACKGROUND

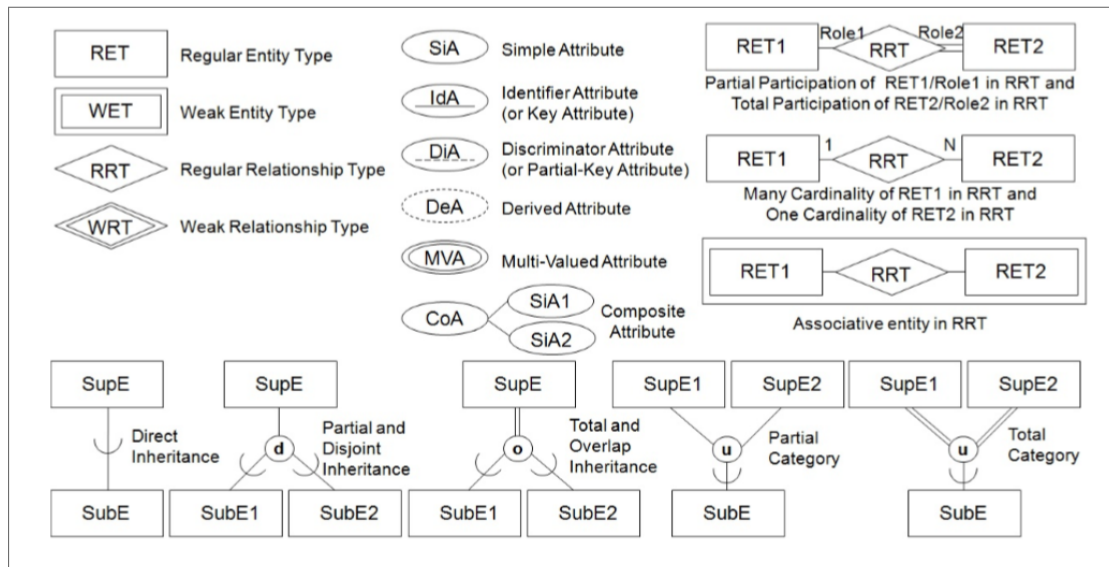
2.1 INTRODUCTION

In the following subsections, we give an overview of the Enhanced Entity-Relationship (EER) model, the Existence Dependency Graphs, the Description Logics, the OWL DL Language, and the Semantic Web Rule Language.

2.2 ENHANCED ENTITY-RELATIONSHIP MODEL

The Enhanced Entity-Relationship Model (EER) is a modeling language for conceptual database design. It is an extension of the Entity-Relationship (ER) model proposed by Chen (1976), including concepts such as "Entity", "Relationship", and "Attribute", and adding the ideas of superclass, subclass, and category, which are semantic data modeling concepts also developed in other areas such as Knowledge Representation and Software Engineering (ELMASRI; NAVATHE, 2010). Due to its relevance and expressiveness, there are several research papers related to the EER language ((ZHANG; MA; CHENG, 2016), (FIDALGO et al., 2013), (VAGNER, 2018), (KOLP; ZIMANYI, 2000)), as well as many database textbooks that present a chapter about the language ((ELMASRI; NAVATHE, 2010), (CONNOLLY; BEGG, 2010), (SINGH, 2009)). Figure 1 shows a summary of the EER notation proposed by Elmasri and Navathe (2010).

Figure 1 – EERMM Notation



From: Fidalgo et al. (2013)

In the EER language, "entity" is a fundamental concept responsible for representing distinctly identifiable objects (CHEN, 1976). These objects can be physical (e.g., person,

car) or conceptual (e.g., companies, services). "Entity types" correspond to schemas or classes responsible for describing the structure of a set of entity objects (or instances). In EER, they are represented as a rectangle, as shown in Figure 1 (Regular Entity Type).

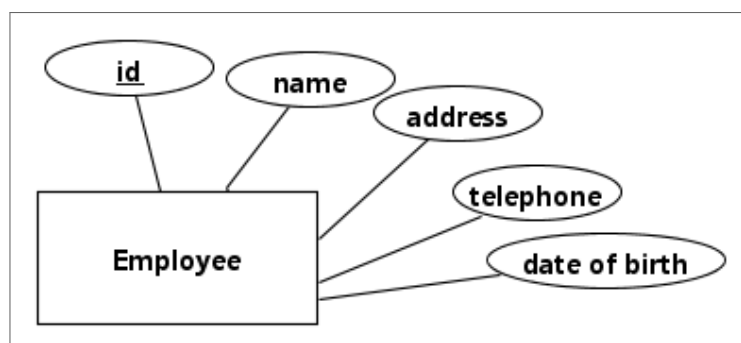
Entity types have "attributes", which are specific properties used to describe entities. In the EER language, they are represented as an ellipse. Figure 2 shows the entity type "Employee" and its attributes: "id", "name", "address", "telephone", and "date of birth".

Attributes can be divided into types. They are:

- **Composite attributes:** Those that can be divided into sub-attributes. In the case of the previous example, the attribute "address" may consist of the sub-attributes "street", "number", "neighborhood", "city" and "state";
- **Simple attributes:** that assume atomic values, that is, do not have sub-attributes;
- **Derived attributes:** Attributes that have values that depend on other attributes. The entity "Employee", for example, could have the attribute "age", derived from "date of birth".
- **Multivalued attributes:** which can assume more than one value for the same entity. The attribute "telephone", for example, can assume more than one value for the same "Employee" instance;

Since an entity is a distinctly identified object, each entity type has one or more attributes whose values do not repeat between their instances. These attributes are called key attributes or identifier attributes. In the "Employee" example, the "id" is the identifier attribute because two employees cannot have the same id value. As shown in Figure 2, identifier attributes have a slightly different notation, with their underlined names.

Figure 2 – Employee entity type and its attributes



From: The author

A "relationship" is an "association among entities" (CHEN, 1976). Similar to what happens with entity types, in the EER model, we have "Relationship types" to represent a set of relationships. Hence, to represent the business rule "an employee manages a

department", we can create the model shown in Figure 3, composed of the entity types "Employee" and "Department," and by the relationship type "Manages".

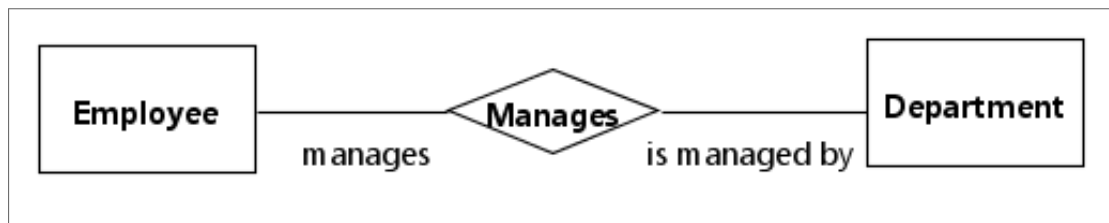
Figure 3 – Relationship example



From: The author

A relationship type can be classified by its degree, the value of which depends on the number of the associated entity types (ELMASRI; NAVATHE, 2010). When a particular relationship type involves only one entity type, it has degree 1, and it is called unary (or recursive). When it associates two entity types, it has degree 2, and it is called binary. Although there may be relationship types with higher degrees, unary, binary, and ternary are the most common ones. Relationship types also have roles that represent the purpose of each entity type in the relationship. In the example shown in Figure 4, the entity type "Employee" has the role "manages" for the relationship type "Manages," while the entity type "Department" has the role "is managed by" for the same relationship type. As highlighted by Elmasri and Navathe (2002), roles are most commonly used in unary relationships.

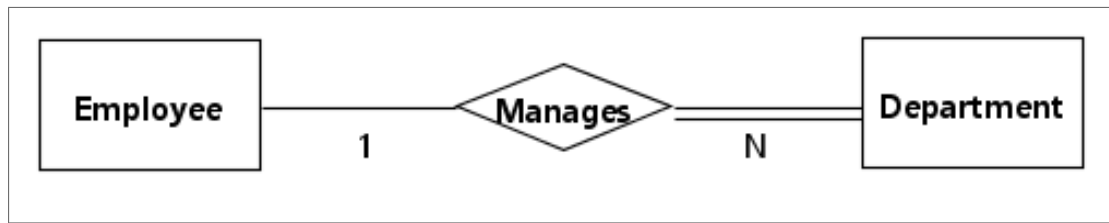
Figure 4 – Model example with roles



From: The author

Relationship types also have structural constraints, which are used to represent certain limits imposed by business rules. They are cardinality and participation. Cardinality constraint refers to the maximum number of instances of an entity type that can participate in a relationship. Cardinality has two possible values: 1 and N (many). The participation constraint determines whether an entity type depends existentially on its participation in a relationship. Participation has two possible values: total (represented by a double line), when the participation is mandatory; and partial (represented by a single line), when the participation is optional. Figure 5 shows an example of a model which represents the business rule "one department must be managed by one employee" and "one employee may manage many departments."

Figure 5 – Model example with cardinality and participation constraints

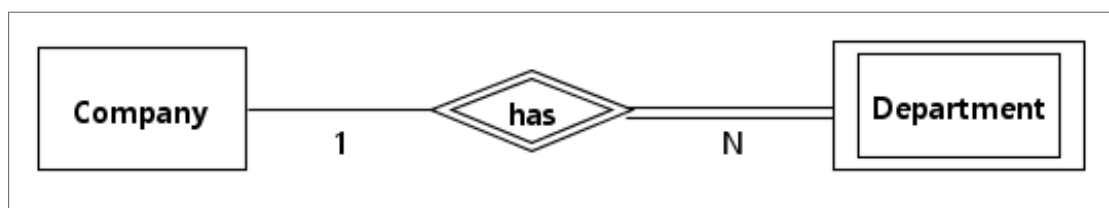


From: The author

Such as entity types, relationship types can also have attributes. In this case, attributes are used to describe properties of relationships. According to Elmasri and Navathe (2002), in 1:1 and 1:N unary and binary relationships, the use of attributes in relationship types is optional because the properties could be moved to one of the participating entities. In unary and binary relationships with N: N cardinality, the use of these attributes is mandatory when it is necessary to describe relationship properties. It is crucial to mention that relationship types cannot have identifier attributes, as this concept is restricted to entity types.

When an entity type does not have identifier attributes, i.e., any of its attribute values can be used to identify its instances uniquely, it is classified as "Weak". Weak entity types are uniquely identified through their relationship to another entity type. In these cases, the relationship type is called "Weak" or "Identifier", and the other entity type is called "Strong". Figure 6 shows a model that represents the rules "one company may have multiple departments" and "one department must belong to one company". In the example, the entity type "Department" is weak in the identifier relationship type "has", while the entity type "Company" is strong. It is important to note that the relationship between the weak entity type and the strong entity type always has total participation, i.e., a weak entity instance depends on the prior existence of a strong entity instance (ELMASRI; NAVATHE, 2010). In the example, there may not be a "Department" instance that is not related to a "Company" instance, since there may not exist a department that is not part of a company. In practice, the weak entity type uses some of its attributes, and the identifier attributes of the strong entity type to be uniquely identified.

Figure 6 – Weak entity type example

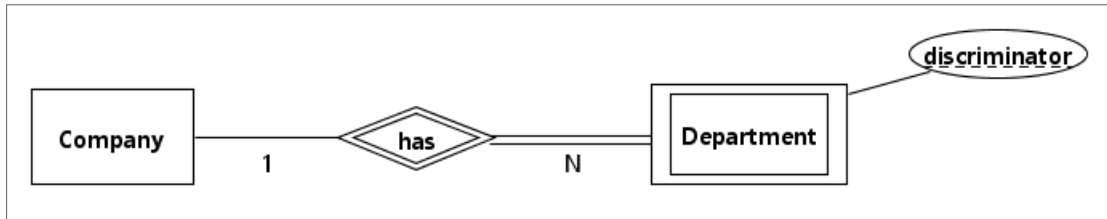


From: The author

The attributes of the weak entity types that are used to compose their key attributes

are called discriminator attributes. These attributes act as partial identifiers, and their values do not repeat within the set of weak entity type instances bound to the same instance of a strong entity type. Discriminator attributes have a notation in which a dotted line underlines their names, as shown in Figure 7. It is important to note that discriminator attributes can also be used to identify the unary and binary relationship types partially when they have N: N cardinality (ELMASRI; NAVATHE, 2010).

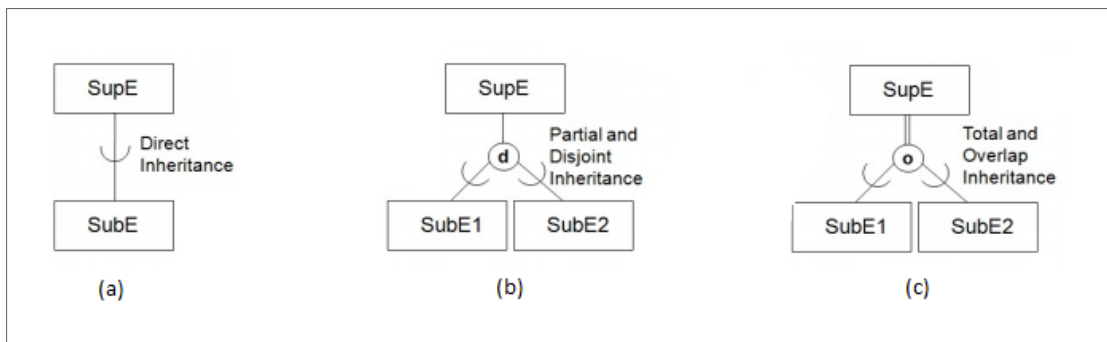
Figure 7 – Weak entity type example with discriminator attribute



From: The author

As mentioned before, the EER language added the concepts of inheritance, category, and associative entity to the ER language. Inheritance (or IS-A relationships) comprises the concepts of superclass and subclass. Thus, the subclass entity type inherits the attributes and relationships of the superclass entity type. The set of subclasses of a given superclass is called a superclass entity type "specialization". On the other hand, we may refer to the superclass as a "generalization" of the set of its subclasses. When a superclass has only one subclass, we represent specialization as a direct inheritance, as can be seen in Figure 8 (a). When a superclass has more than one subclass, the notation used is shown in Figure 8 (b) and (c).

Figure 8 – Inheritance notations



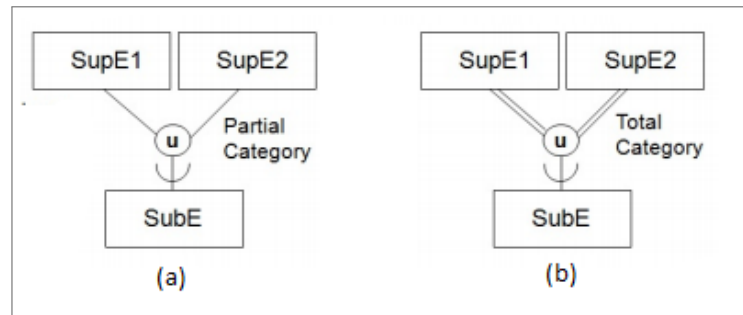
From: Fidalgo et al. (2013)

There are two types of constraints to apply to the relationship between superclasses and subclasses: disjunction and completeness. Disjoint and overlap are types of disjunction constraints. In Figure 8 (b) we have the representation of a disjoint specialization, indicated by the letter "d" (disjoint), which features inheritances in which an entity may be a member of at most one of the subclasses (ELMASRI; NAVATHE, 2010). Figure 8 (c)

shows the representation of an overlapping specialization, represented by the letter "o" (overlap), which characterizes inheritances in which the same entity can be a member of more than one subclass of the specialization. The completeness constraints are "total" and "partial". A total constraint defines that every entity with the superclass type must be a member of some subclass in the specialization (ELMASRI; NAVATHE, 2010). Partial restriction, on the other hand, allows an entity not to belong to any of the subclasses (ELMASRI; NAVATHE, 2010).

The category is a concept used when it is necessary to model scenarios in which a subclass has multiple superclasses. In this case, the subclass is called a category, which can be total or partial. In a total category, the entity set of the subclass corresponds to the union of all superclasses entities (Figure 9 (b)). In a partial category (Figure 9 (a)), the entity set of the subclass is a subset of the union of the superclass entities.

Figure 9 – Inheritance notations



From: Fidalgo et al. (2013)

Finally, the Associative Entity is an abstraction used when it is necessary to group a set of entities and relationships to which other relationships and entities can be associated. Figure 1 shows a representation of an associative entity (Associative Entity in RRT).

2.3 EXISTENCE DEPENDENCY GRAPHS

According to Snoeck (2014), the concept of Existence Dependency can be defined based on two levels of abstraction: the class level and the object level. When we consider two classes A and B, we say that B is existentially dependent on A ($B \leftarrow A$), if each type B object is necessarily associated with at least one, at most one, and the same type A object. Class B is called "dependent" and class A is called "master".

In conceptual modeling, it is not uncommon to come across cases of existence dependency. In the EER language, for example, the relationship between weak and strong entity types is an existence dependency relationship, in which the weak one is existentially dependent on the strong one.

Snoeck (2014) also defined that existence dependency relationships must not be part of cyclic paths within a model. A path refers to the connectivity of entities and relationships.

They are responsible for defining "the structural association that each entity has simultaneously with all other entities or with itself within the path" (DULLEA; SONG; LAMPROU, 2003). Based on that, there are two types of paths: cyclic, i.e., a path that recurs back to a previous entity, and acyclic, i.e., when there are no cycles. Existence dependency relationships must not occur in a cyclic path because of the following restrictions: "an object type is never existence dependent on itself" (SNOECK, 2014), and "the master of an object type can never be a dependent (directly or indirectly) of that same object type" (SNOECK, 2014).

2.4 DESCRIPTION LOGICS

Description Logic (DL) is a set of languages used for knowledge representation. Through DL it is possible to describe three types of entities: concepts, which represent a set of individuals (object classes); roles, which represent binary relationships between individuals; and individual names, which represent individuals who are part of a particular domain (KROTZSCH; SIMANCIK; HORROCKS, 2012).

Description Logic allows the creation of Knowledge Bases (KBs) that consist of three components: a TBox, an ABox, and a RBox (KROTZSCH; SIMANCIK; HORROCKS, 2012). The TBox describes the vocabulary of a particular domain through axioms (statements that may be true according to the context), while the ABox contains statements about individuals in the domain, for example by determining how they relate and to which concepts they belong. The RBox, in turn, describes the properties of relations, such as disjunction or equivalence between roles.

With Knowledge Bases, it is possible to perform reasoning tasks using specific tools called reasoners. By means of these tools it is possible, for example, to determine if a KB is satisfiable, i.e., it has no contradictions, and to establish hierarchies between concepts, automatically identifying which are more general (BAADER, 2003).

The reasoning capability of the reasoners varies according to the expressiveness of the chosen description logics language.

2.4.1 \mathcal{AL} Language

As mentioned, Description Logics expressivity varies according to the available constructors of its languages. The \mathcal{AL} language is presented in the literature as a minimal language for use in practical applications (BAADER, 2003). It is used as the basis for the extension of a language family. Table 1 shows the syntax and semantics of its constructors, considering "R" as a generic binary relation.

With a DL language, it is possible to describe atomic concepts and atomic roles, as well as complex descriptions (axioms). In Table 1, the atomic concept "Student" describes the set of individuals that belong to the "Student" class, while the atomic con-

Table 1 – \mathcal{AL} language constructors.

Constructor	Syntax	Semantics	Examples
Atomic Concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$Student$
Atomic Negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	$\neg Student$
Top Concept	\top	$\Delta^{\mathcal{I}}$	\top
Bottom Concept	\perp	\emptyset	\perp
Intersection	$C \sqcap D$	$(C^{\mathcal{I}} \cap D^{\mathcal{I}})$	$Student \sqcap Woman$
Value Restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	$\forall isEnrolled.Subject$
Limited Existential Quant.	$\exists R.\top$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in R^{\mathcal{I}}\}$	$\exists isEnrolled.\top$

Adapted from: Baader (2003)

cept "Woman" describes all individuals that belong to the "Woman" class. An example of a complex description would be $FemaleStudent \sqsubseteq Student \sqcap Woman$, which represents the set of "all students who are also women" through the use of the conjunction operator (\sqcap). Another example would be the description $PostGradStudent \equiv Student \sqcap \exists isEnrolled.PostGradCourse$, which represents "all students enrolled in a postgraduate course".

In the descriptions presented, we used different symbols for the definition of axioms: \sqsubseteq and \equiv . When we have $A \sqsubseteq B$, the symbol \sqsubseteq represents a primitive definition, determining that B describes the conditions necessary for a group of individuals to be represented by the concept A, i.e., $B \rightarrow A$. When we have $A \equiv B$, the symbol \equiv represents the necessary and sufficient conditions for a set of individuals to be represented by the concept A. In other words, $A \equiv B$ can be read as "A equals B", i.e., $A \sqsubseteq B$ and $B \sqsubseteq A$ (HORRIDGE, 2011).

In \mathcal{AL} language, it is also possible to describe the negation of an atomic concept. Hence, considering the Student atomic concept defined above, the description $\neg Student$ includes all individuals who do not belong to the Student class. When we need to represent all individuals in a given Knowledge Base, we can use the universal concept, represented by the symbol \top . On the other hand, when we need to represent a concept that does not comprise any individual, we can use the empty concept, represented by the symbol \perp . Also, in the \mathcal{AL} language, only the universal concept is allowed to determine the scope of a limited existential quantification ($\exists R.\top$, where R is an atomic relation), as shown in Table 1 (BAADER, 2003).

As shown in Table 1, each constructor is associated with formal semantics, represented by the $.^I$ (interpretation) function. Thus, for the atomic concept A, the interpretation function A^I consists of its set of possible interpretations, while the set of all possible interpretations of a domain is represented by the function Δ^I . The formal semantics associated with each \mathcal{AL} language constructor is described in Table 1.

2.4.2 *ALCROIQ* Language

As commented in the previous subsection, the \mathcal{AL} language is used as the basis for a language family whose members differ in their expressive power according to the allowed constructors. The *ALCROIQ* language makes use of the additional constructors presented in Table 2, having the level of expressiveness required for the logical description of the EER language, presented in the next chapter.

According to Table 2, it can be noted that the letter corresponding to disjunction (\sqcup) is omitted from the language name. According to De Morgan's laws, through conjunction (\sqcap) and negation (\neg) constructors, it is possible to describe disjunctions (\sqcup).

Table 2 – *ALCROIQ* language constructors.

Constructor	Letter	Syntax	Semantics
Disjunction	\mathcal{U}	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Concept Negation	\mathcal{C}	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Role Disjunction	\mathcal{R}	$Disj(S_1, S_2)$	$S_1^{\mathcal{I}} \cap S_2^{\mathcal{I}} = \emptyset$
Role Inclusion	\mathcal{R}	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Named Individuals	\mathcal{O}	$\{\text{mary}\}$	$\{\text{mary}^{\mathcal{I}}\}$
Inverse Roles	\mathcal{I}	R^-	$\{(a, b) (b, a) \in R^{\mathcal{I}}\}$
Qualified Value Restriction	\mathcal{Q}	$(\geq kR.C)$ $(\leq kR.C)$	$\geq \{x \#\{(y.(x, y) \in R^{\mathcal{I}}) \text{ and } (y \in C^{\mathcal{I}}) \geq k\}$ $\leq \{x \#\{(y.(x, y) \in R^{\mathcal{I}}) \text{ and } (y \in C^{\mathcal{I}}) \leq k\}$

Adapted from: Baader (2003)

2.5 OWL DL AND THE SEMANTIC WEB RULE LANGUAGE

According to Bechhofer et al. (2004), the Web Ontology Language (OWL) is "a semantic markup language for publishing and sharing ontologies on the World Wide Web".

OWL consists of three sublanguages (BAADER; HORROCKS; SATTLER, 2004):

- OWL Full: which corresponds to the full expressivity and power of the language. It is undecidable, i.e., does not provide complete reasoning support;
- OWL DL: a decidable subset of OWL Full. Its semantics is based on Description Logics;
- OWL Lite: which corresponds to a restricted (less expressive) version of OWL DL.

In this work, we used Protégé, an ontology editor (MUSEN, 2015), to create an ontology using the OWL DL language. Since OWL DL semantics is based on Description Logics, we could translate the OWL DL ontology created into a DL Knowledge Base, as will be presented in chapter 4.

The Semantic Web Rule Language (SWRL) is a union of the OWL DL and OWL Lite with sublanguages of the Rule Markup Language (HORROCKS et al., 2004). SWRL makes it possible to reason about OWL individuals (ABox) by defining rules. Hence, in order to express that "two people are cousins when their parents are siblings" we could use the rule shown in Figure 10. In this case, the OWL ontology must define the concept of "Person" and the roles "siblings", "parentOf", and "cousins".

Figure 10 – SWRL Rule Example

```
Person(?x1) ^ Person(?x2) ^ Person(?x3) ^  
Person(?x4) ^ differentFrom(?x1,?x2) ^  
differentFrom(?x1, ?x3) ^ differentFrom(?x1,?x4) ^  
differentFrom(?x2, ?x3) ^ differentFrom (?x2, ?x4)  
^ differentFrom(?x3, ?x4)^  
parentOf(?x1, ?x2) ^  
parentOf(?x3, ?x4) ^ siblings(?x1, ?x3)  
-> cousins(?x2, ?x4)
```

From: The author

3 EER SEMANTIC RULES

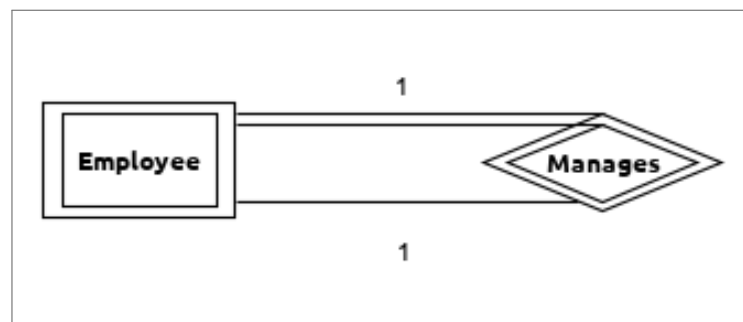
3.1 INTRODUCTION

In this chapter, we briefly present the Enhanced Entity-Relationship semantic rules considered in this work for the automatic validation of EER models.

3.1.1 Validity Rules Catalog

As mentioned in chapter 1, to create conceptual models, designers make use of modeling languages. Modeling languages comprises three elements: abstract syntax, concrete syntax, and semantics. Therefore, when validating models, it is necessary to take into account syntactic and semantic errors. It is also important to note that a model can be syntactically valid but semantically invalid. The EER model presented in Figure 11, for example, although syntactically valid (uses the correct notation for each concept represented and connects the elements correctly), presents a semantic error. As stated by Nascimento (2015), identifier relationship types must not be unary. In the case of a unary identifier relationship type, the entity type would act as strong and weak in the same association, i.e., it would be existentially dependent on itself. Since the concept of existence dependency does not allow such circumstances, the model is semantically invalid.

Figure 11 – Semantically invalid model



From: The author

To prevent semantically invalid conceptual models from being used as a basis for database projects, Nascimento (2015) proposed to organize a catalog of validity rules for EER models.

The catalog consists of rules defined by Dullea, Song and Lamprou (2003) and Calvanese and Lenzerini (1994), as well as rules defined by Nascimento (2015) using the following methodology:

1. For each relationship type (Regular and Identifier) and degree (Unary, Binary, and Ternary), he analyzed all possible links with the associative entity type and each

entity type (regular, strong, weak, superclass, and subclass);

2. Next, he analyzed all possible combinations of generalization and specialization relationships between entity types;
3. After that, he focused on attribute relationships, by verifying all possible links between the attribute types, each entity type, and each relationship type and degree.
4. Finally, the analysis consisted of identifying structurally invalid combinations, involving the constructors of the EER model from (1), (2) and (3) and the constraints of cardinality, participation in cyclic and acyclic paths.

As commented in chapter 1, the catalog presents limitations. The methodology used to obtain the rules analyzed a set of constructors and their interactions in isolation and did not evaluate the consequences of the interactions between multiple sets of constructors in the same model. Also, the rules do not cover relationship type degrees above 4, nor the category constructor, because of their restricted use. Despite these limitations, Nascimento (2015) catalog provided excellent coverage of EER language constructors and was considered in this work for the automatic validation of EER models.

Table 3 shows the catalog. The author named the rules according to the following pattern: "R" (Rule) + a character representing the constructor for the rule (unary relationship type (U), binary relationship type (B), ternary relationship type (T), attribute type (A) and inheritance (H)) + [01-99] (rule order number). To make explicit the total amount of rules, we renamed them according to the following pattern: R + [00-99], in which "R" corresponds to the word "rule", and the numbering corresponds to the rule id in the catalog.

It is essential to mention that, although the catalog consists of 28 rules, we only formalized 27 of them since the rule RA01, that states that "regular entities must have identifier attributes", could not be caught by DL reasoners because of the open-world assumption. According to Krotzsch, Simancik and Horrocks (2012), the open-world assumption "keeps unspecified information open", i.e., if a fact is not explicitly specified (e.g., the presence or absence of an identifier attribute), it cannot be assumed to be true or false.

In the next sections, we will present all the rules, grouping them according to the methodology used by Nascimento (2015) (unary relationship types, binary relationship types, ternary relationship types, inheritance, and attribute types).

Table 3 – EER structural validity rules catalog

Our code	Nascimento (2015) code	Description
----------	------------------------	-------------

R01	RU01	All 1:1 unary relationship types with total-partial or partial-total participation are structurally invalid.
R02	RU02	All 1 : M or M : 1 unary relationship types with total-total participation are structurally invalid.
R03	RU03	All 1 : M or M : 1 unary relationship types with total participation on the 'one' side and partial participation on the 'many' side are structurally invalid.
R04	RU04	Identifier relationship types must not be unary.
R05	RB01	Cyclic paths containing no opposing and no self-adjusting binary relationship types are structurally invalid.
R06	RB02	Associative entities must not have total participation in relationships.
R07	RB03	No strong entity type link in an identifier relationship type can have cardinality "N".
R08	RB04	An identifier relationship type can not be doubly identified.
R09	RB05	Identifier relationship types are not allowed to form a cyclic path.
R10	RB06	The identifying links between a weak entity type and an identifier relationship type must have "total" participation.

R11	RT01	Cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the cardinality constraints of the binary relationship type are less than the cardinality constraints of the ternary one.
R12	RT02	Cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the participation constraints of the binary relationship type are less than the participation constraints of the ternary one.
R13	RT03	A relationship type must not be ternary and identifier at the same time.
R14	RH01	A model must not have inheritances forming a cyclic path.
R15	RH02	An entity type must not be weak and a subclass in the same model.
R16	RH03	Superclass entity types must not be weak entities of its subclasses.
R17	RH04	Models that present relationship types that result in restrictions on the number of instances different from those imposed by the inheritances are structurally invalid.
R18	RH05	Associative entities must not participate in inheritances.
-	RA01	Regular entities must have identifier attributes.
R19	RA02	Regular entity types should not have a discriminator attribute.

R20	RA03	Weak entity types must not have identifier attributes.
R21	RA04	Weak entity types must not have discriminator attributes if the identifier relationship type has 1:1 cardinality.
R22	RA05	If a weak entity type participates in more than one identifier relationship, they all must have "1:N" cardinality.
R23	RA06	Associative entities can not contain attributes.
R24	RA07	Subclass entity types can not have identifier attributes.
R25	RA08	Subclass entity types can not have discriminator attributes.
R26	RA09	Relationships types can not have identifier attributes.
R27	RA10	Only unary and binary relationship types with cardinality "N:N" may have discriminator attributes.

Adapted from: (NASCIMENTO, 2015)

3.1.2 Unary Relationship Type

R01 to R04 refers to unary relationship types associations to entity types and associative entity.

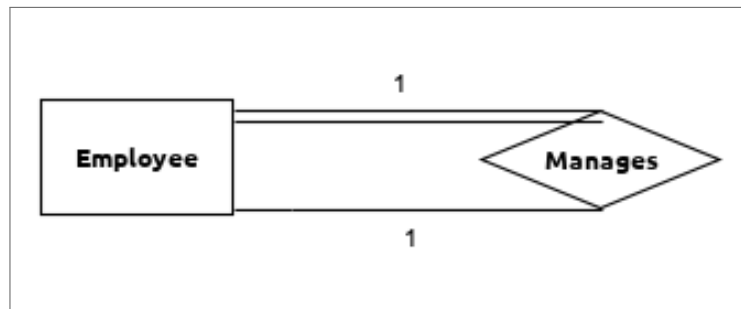
R01 to R03 are rules based on the results found in Dullea, Song and Lamprou (2003). After analyzing all the combinations between cardinality and participation constraints, the authors presented the following corollaries:

- all 1:1 unary relationship types with total-partial or partial-total participation are structurally invalid (R01);
- all 1 : M or M : 1 unary relationship types with total-total participation are structurally invalid (R02); and

- all 1 : M or M : 1 unary relationship types with total participation on the ‘one’ side and partial participation on the ‘many’ side are structurally invalid (R03).

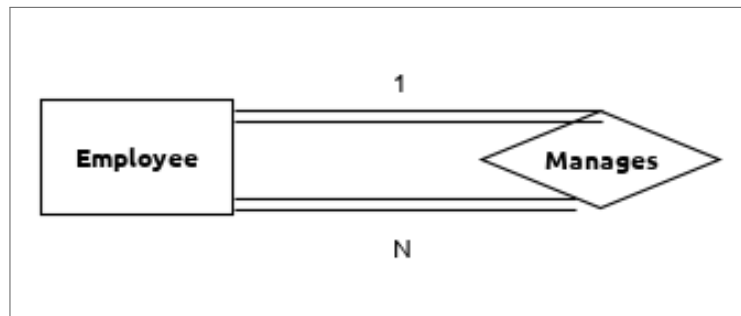
The authors proved that all remaining combinations are valid. Figures 12, 13, and 14 shows examples of invalid models according to R01, R02, and R03, respectively. Although the examples only show regular entity types, the corollaries also apply to associative entities and the remaining entity types, since they assume the same role of a regular entity type in unary relationships.

Figure 12 – Invalid model according to R01



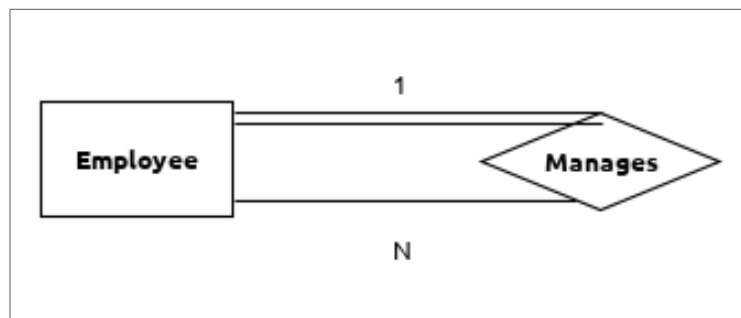
From: The author

Figure 13 – Invalid model according to R02



From: The author

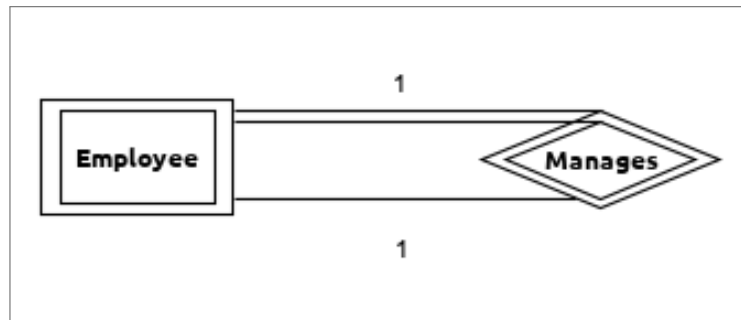
Figure 14 – Invalid model according to R03



From: The author

R04 states that identifier relationship types must not be unary. The rule was defined by Nascimento (2015) and is related to the existence dependency between the weak entity type and the strong entity type, as discussed in chapter 2. In the case of a unary identifier relationship type, the entity type would act as strong and weak in the same association, i.e., it would be existentially dependent on itself. Since the concept of existence dependency does not allow such circumstances, the model is structurally invalid. Figure 15 shows an invalid model, according to R04.

Figure 15 – Invalid model according to R04



From: The author

3.1.3 Binary Relationship Type

R05 to R10 refers to binary relationship types.

R05 refers to the existence of binary relationship types, forming a cyclic path and is based on a corollary defined by Dullea, Song and Lamprou (2003).

In order to understand the rule, consider that $|E|$ is the relative number of instances an entity type "E" can have when participating in a relationship. Since, in binary relationship types, two entity types are associated by one relationship, there are three possible restrictions on the number of instances between them: $|E_1| = |E_2|$, $|E_1| > |E_2|$ and $|E_1| < |E_2|$. Relationships that support all cases are called "self-adjusting". When we have a cyclic path that contains only binary relationship types and one relationship type states that $|E_1| > |E_2|$ and another one states that $|E_2| < |E_1|$, we say that those relationship types complement each other and we call them "opposing relationship types". Table 4 shows the restrictions on the number of instances between entity types for all the possible combinations of cardinality and participation in binary relationship types. Based on this analysis, Dullea, Song and Lamprou (2003) stated that binary relationship types involved in acyclic paths are always structurally valid, but cyclic paths containing no opposing and no self-adjusting relationship types are structurally invalid (R05).

Figure 16 shows an example of a structurally invalid model according to R05. It states that:

1. $|Lecturer| < |Course|$;

2. $|Course| < |Project|$;
3. $|Project| < |Lecturer|$.

By transitivity, we have that $|Course| < |Lecturer|$, $|Project| < |Course|$, and $|Lecturer| < |Project|$, which contradicts (1), (2), and (3), respectively, and thus results in an invalid model.

Table 4 – Structural restrictions of a binary relationship.

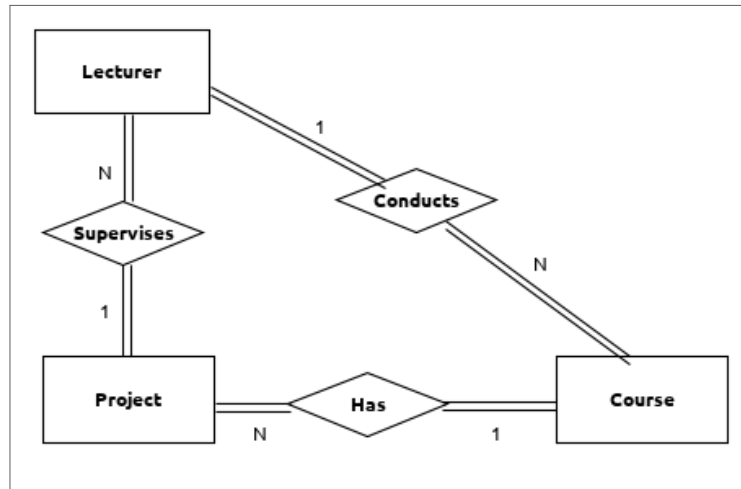
Case	Cardinality	Participation	Restrictions
1	1:1	Total-Total	$ E_1 = E_2 $
2	1:1	Total-Partial	$ E_1 < E_2 $
3	1-1	Partial-Total	$ E_1 > E_2 $
4	1:1	Partial-Partial	Self-adjusting
5	N:1	Total-Total	$ E_1 > E_2 $
6	N:1	Total-Partial	Self-adjusting
7	N:1	Partial-Total	$ E_1 > E_2 $
8	N:1	Partial-Partial	Self-adjusting
9	1:N	Total-Total	$ E_1 < E_2 $
10	1:N	Total-Partial	$ E_1 < E_2 $
11	1:N	Partial-Total	Self-adjusting
12	1:N	Partial-Partial	Self-adjusting
13	N:N	Total-Total	Self-adjusting
14	N:N	Total-Partial	Self-adjusting
15	N:N	Partial-Total	Self-adjusting
16	N:N	Partial-Partial	Self-adjusting

Adapted from: Dullea, Song and Lamprou (2003)

R06 states that associative entities must not have total participation in relationships. Associative Entities are constructs used when we need to create a relationship between two relationship types. According to Nascimento (2015), when we have an associative entity with total participation in a relationship, we are representing a ternary relationship between the entity types involved. Figure 17 shows an example of a structurally invalid model, according to R06. It implies a ternary relationship between "Team", "Project", and "Budget", since the total participation forces entity types "Team" and "Project" to participate in the "Has" relationship type, and therefore the use of the associate entity notation makes the diagram structurally invalid.

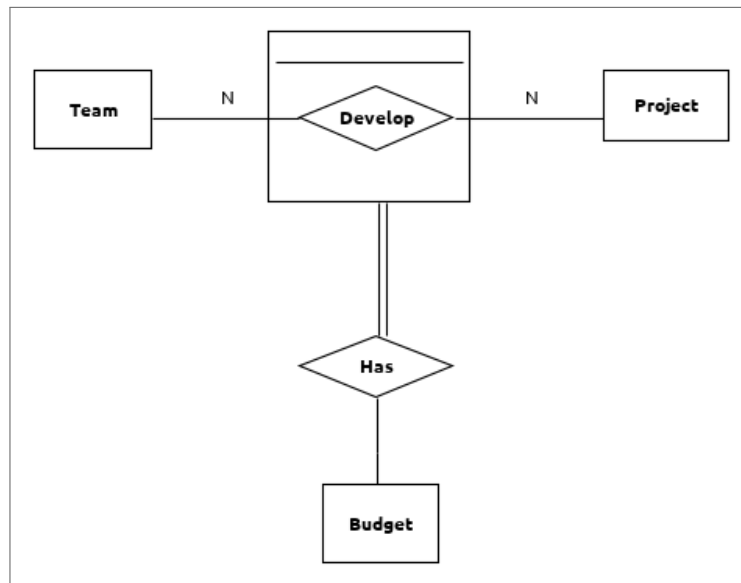
R07 determines that no strong entity type link in an identifier relationship type can have cardinality "N". As mentioned before, an instance of a weak entity type depends on the previous existence of a strong entity instance. When we have a cardinality "N", we

Figure 16 – Invalid model according to R05



From: The author

Figure 17 – Invalid model according to R06

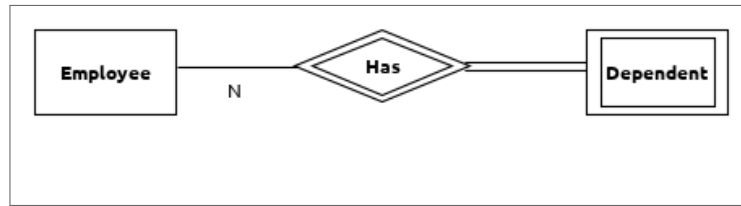


From: The author

state that one weak entity type instance depends on the previous existence of more than one strong entity type instance which implies in a structurally invalid model. Figure 18 shows an example of a structurally invalid model, according to R07.

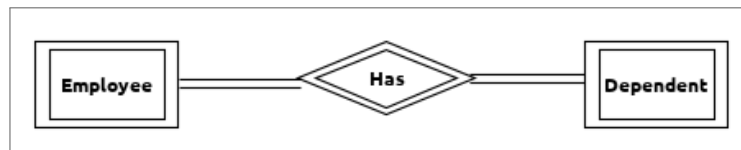
R08 states that an identifier relationship type can not be doubly identified. A doubly identified relationship implies the existence of two weak entity types that depend on each other. As previously mentioned, an instance of a weak entity type depends on the previous existence of a strong entity instance. Since in a doubly identified relationship, we only have weak entity types that depend on each other, we could never instantiate them, which implies in an invalid model. Figure 19 shows an example of a structurally invalid model, according to R08.

Figure 18 – Invalid model according to R07



From: The author

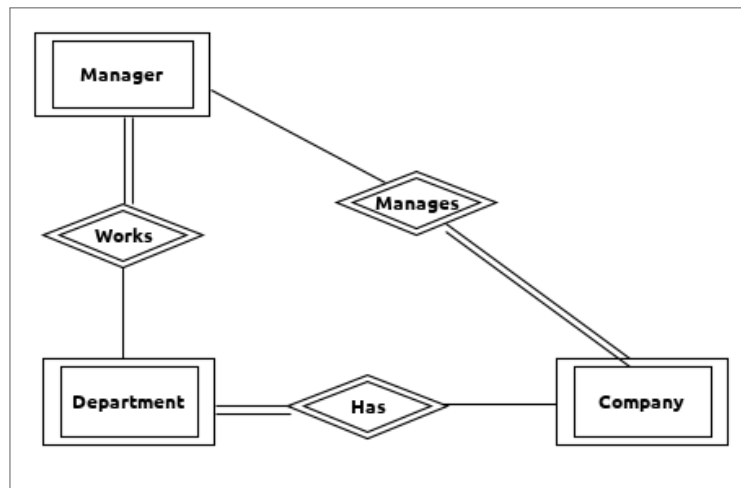
Figure 19 – Invalid model according to R08



From: The author

R09 declares that identifier relationship types are not allowed to form a cyclic path. As discussed in chapter 2, in an identifier relationship type, the weak entity type is existentially dependent on the strong entity type. Since existence dependency is not allowed in cyclic paths, a model that contains identifier relationship types forming a cyclic path is structurally invalid. Figure 20 shows an example of an invalid model, according to R09.

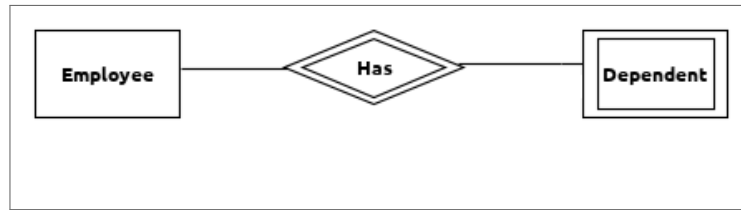
Figure 20 – Invalid model according to R09



From: The author

R10 states that all identifying links between a weak entity type and an identifier relationship type must have "total" participation. The existence of an identifier link with "partial" participation implies that there may be instances of the weak entity type that do not depend on the existence of a strong entity type instance, which contradicts the concept of weak entity. Figure 21 shows an example of a structurally invalid model, according to R10.

Figure 21 – Invalid model according to R10



From: The author

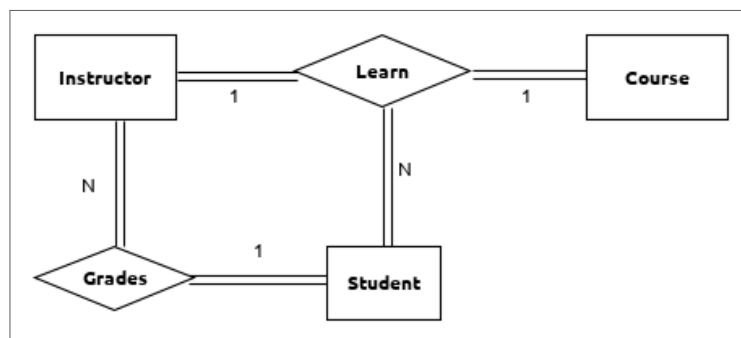
3.1.4 Ternary Relationship Type

R11 to R13 refer to structural validity rules related to ternary relationship types.

R11 and R12 were based on the corollaries found in Dullea, Song and Lamprou (2003). After analyzing models containing ternary relationship types, as well as cardinality and participation constraints imposed on the ternary relationship types, the authors stated that:

- acyclic paths that contain ternary relationship types are always structurally valid;
- cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the cardinality constraints of the binary relationship type are less than the cardinality constraints of the ternary one (R11) (Figure 22).
- cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the participation constraints of the binary relationship type are less than the participation constraints of the ternary one (R12) (Figure 23).

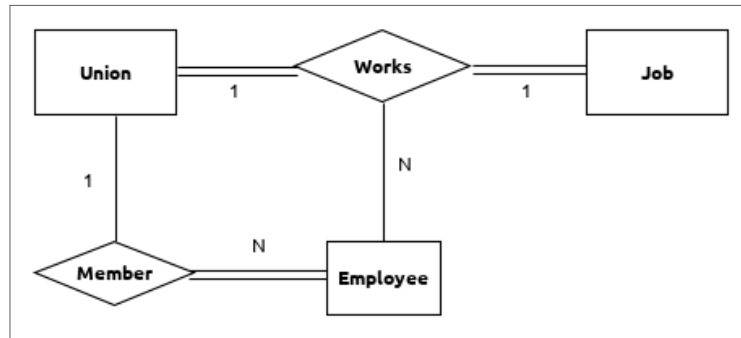
Figure 22 – Invalid model according to R11



From: The author

A binary relationship type constrains a ternary relationship type when two of the three entity types that participate in the ternary relationship also participate in the binary one, and the binary one is directly related to the scenario covered by the ternary one. According to Dullea, Song and Lamprou (2003), there are real-world scenarios that

Figure 23 – Invalid model according to R12



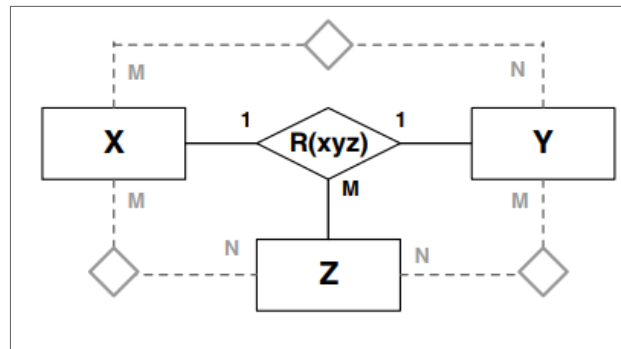
From: The author

can not be modeled by using just a ternary relationship type, and that is why we might need to add supplementary relationship types.

As discussed by Dullea, Song and Lamprou (2003), all ternary relationships types implicitly imply in "N:N" binary relationships between any two Entities that participate in them (Figure 24).

In the example shown in Figure 25 (a), we have the entity types "Budget", "Team", and "Project", that participate in the ternary relationship type "Works". It also shows the implicit binary relationship types. According to Dullea, Song and Lamprou (2003), in order to represent the fact that "Projects can only be funded from a single budget", it would be necessary to add a binary relationship type between "Projects" and "Budget" (Figure 25 (b)). Hence, the binary relationship type is directly related to the scenario covered by the ternary one, i.e., it constrains the ternary relationship type and must follow R11 and R12 in order to be valid. Unfortunately, the EER language has no constructors to indicate when a binary relationship type constrains a ternary relationship type. Therefore, structural errors according to R11 and R12 must be checked against business rules.

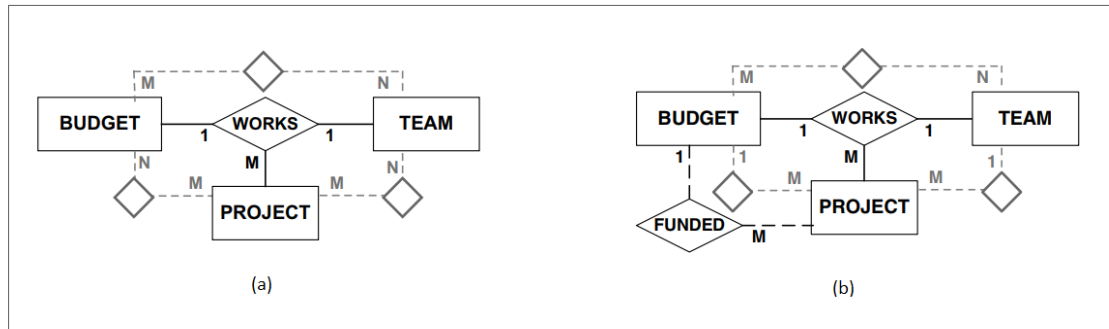
Figure 24 – Implicit binary relationships in ternary relationships



From: Dullea, Song and Lamprou (2003)

R13 asserts that a relationship type must not be ternary and identifier at the same time. As shown in Figures 26 and 27, when we convert an identifier ternary relation-

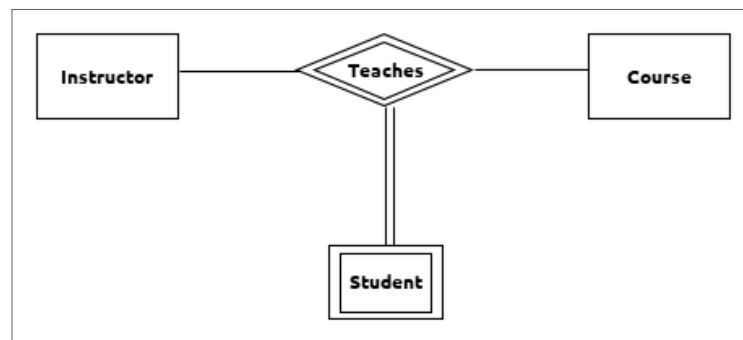
Figure 25 – Ternary relationship example



From: Dullea, Song and Lamprou (2003)

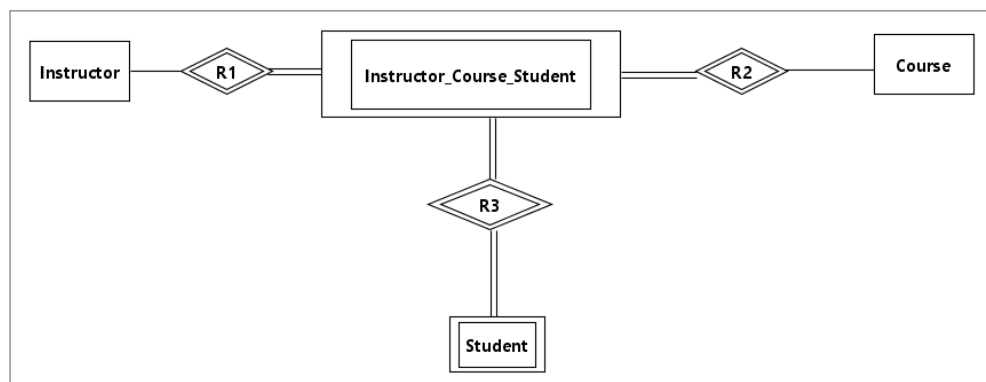
ship type to its binary equivalent, we find a structurally invalid relationship type (R3) according to the R08.

Figure 26 – Invalid model according to R13



From: The author

Figure 27 – Binary equivalent model



From: The author

3.1.5 Inheritance

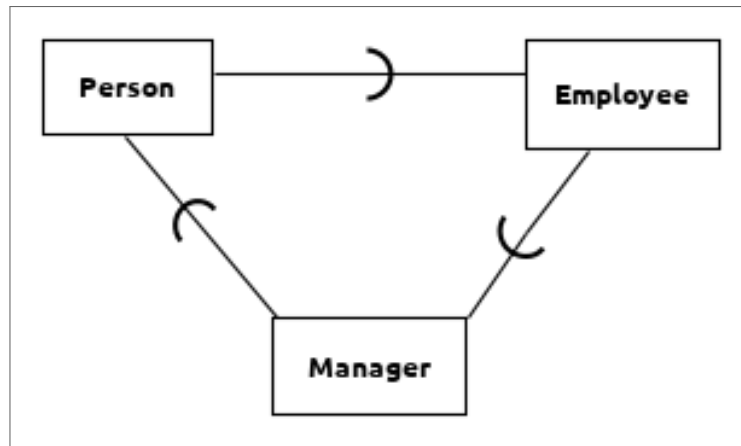
R14 to R18 refers to inheritances.

R14 defines that a model must not have inheritances forming a cyclic path. Figure 28 shows an example of an invalid model, according to R14. Inheritance implies that the number of instances of a superclass entity type must be equal or greater than the number of instances of its subclasses (CALVANESE; LENZERINI, 1994). Performing an analysis similar to the one made by Dullea, Song and Lamprou (2003), the model shown in Figure 28 states that:

1. $|Person| \geq |Employee|$;
2. $|Employee| \geq |Manager|$;
3. $|Manager| \geq |Person|$;

By transitivity, we have that $|Person| \geq |Person|$, $|Employee| \geq |Employee|$, and $|Person| \geq |Person|$, which is invalid since the same entity set can not be greater than itself.

Figure 28 – Invalid model according to R14

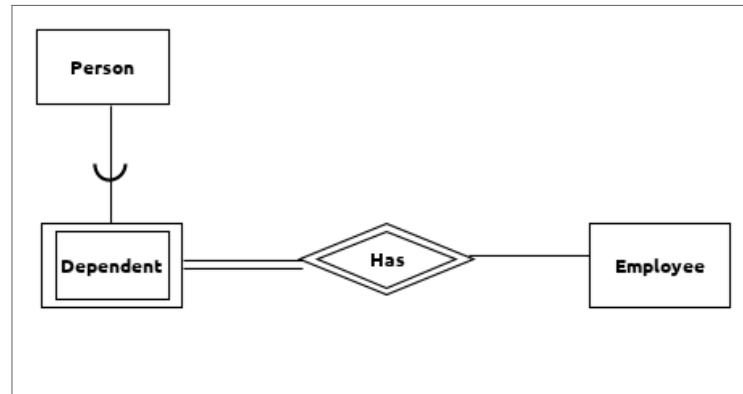


From: The author

R15 states that an entity type must not be weak and a subclass in the same model. As previously mentioned, a weak entity type does not have identifier attributes and is uniquely identified by its relationship with a strong entity type. On the other hand, subclass entity types are uniquely identified by the values of their attributes. If an entity type is a subclass, it could not be a weak entity type, since it would be identified by its attribute values instead of by its relationship with a strong entity type. Also, a weak entity type could not be a subclass, since it would be already uniquely identified by its relationship with a strong entity type. Figure 29 shows an example of a structurally invalid model, according to R15.

R16 defines that superclass entity types must not be weak entities of its subclasses. In these cases, the model presents a cyclic path in which the superclass is existentially dependent on its subclasses, and also the subclasses are existentially dependent on the

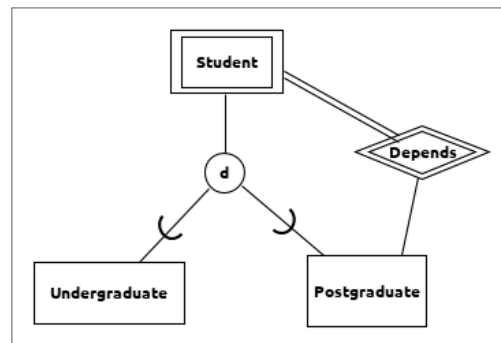
Figure 29 – Invalid model according to R15



From: The author

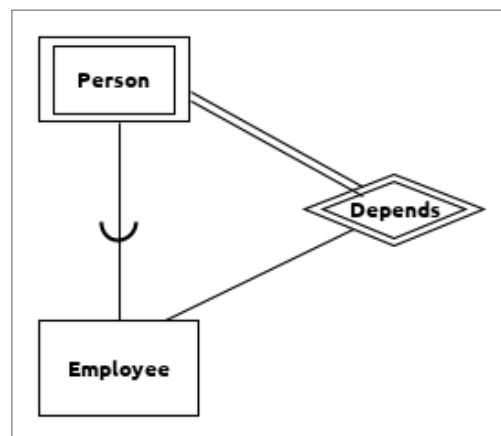
superclass. Since existence dependency is not allowed in cyclic paths, the construct is structurally invalid. Figures 30 and 31 shows an example of a structurally invalid model, according to R16.

Figure 30 – Invalid model according to R16



From: The author

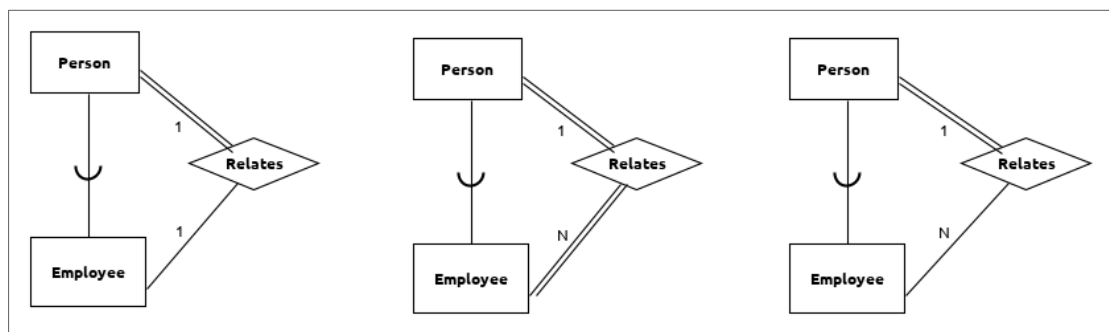
Figure 31 – Invalid model according to R16



From: The author

R17 states that models that present relationship types that result in restrictions on the number of instances different from those imposed by the inheritances are structurally invalid. Since inheritance implies that the number of instances of a superclass entity type must be equal or greater than the number of instances of its subclasses (CALVANESE; LENZERINI, 1994), we can not have a binary relationship type between those two entity types that contradicts the condition. We could refer to the work of Dullea, Song and Lamprou (2003) to check all cardinality and participation constraints combination that would make the model structurally invalid. Figure 32 show examples of invalid models according to R17.

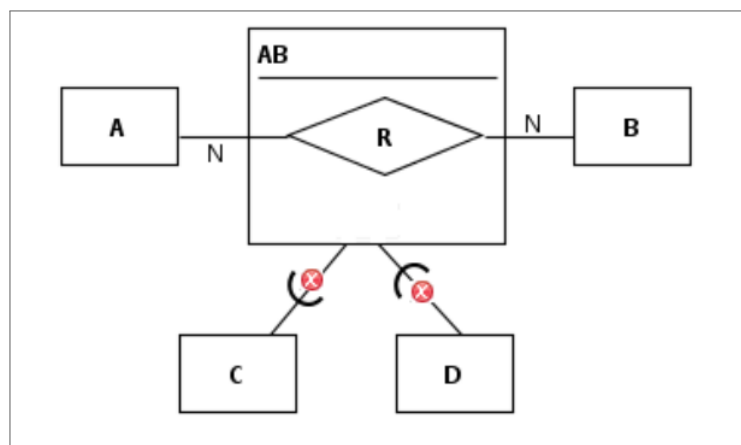
Figure 32 – Invalid models according to R17



From: The author

R18 determines that associative entities must not participate in inheritances. According to Nascimento (2015), the associative entity is a constructor created only for making it possible to link two relationship types, and therefore should not participate in inheritances. Figure 33 shows an example of a structurally invalid model, according to R18.

Figure 33 – Invalid model according to R18



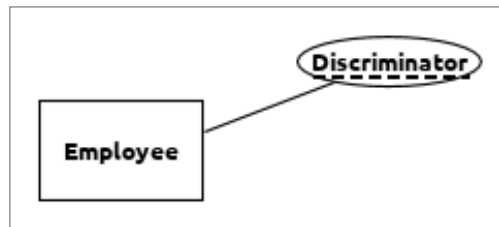
From: Nascimento (2015)

3.1.6 Attribute

R19 to R27 refers to attribute validity rules.

R19 asserts that regular entity types should not have a discriminator attribute. Regular entity types comprise entity types that are uniquely identified by its identifier attributes. Thus the concept comprises entity types that do not participate in identifier relationships and those that act as strong entity types in all the identifier relationships in which they participate. As discussed in chapter 2, discriminator attributes act as partial identifiers. Since regular entity types have identifier attributes, they do not need partial identifiers. Figure 34 shows an invalid model, according to R19.

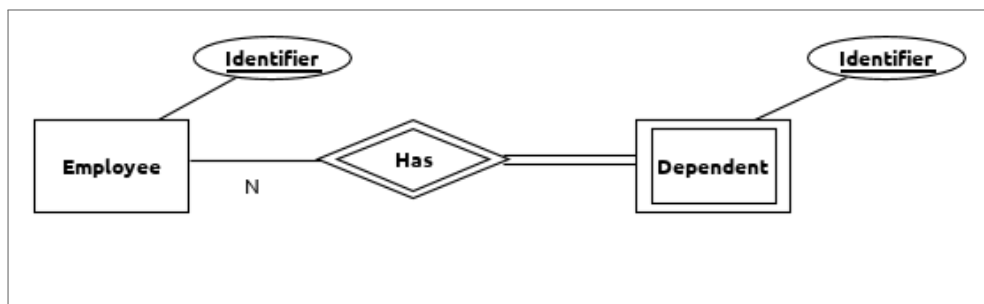
Figure 34 – Invalid model according to R19



From: The author

R20 states that weak entity types must not have identifier attributes. As discussed in chapter 2, weak entity types are identified by their relationship with a strong entity type; consequently, they do not have identifier attributes. Figure 35 shows an invalid model, according to R20.

Figure 35 – Invalid model according to R20

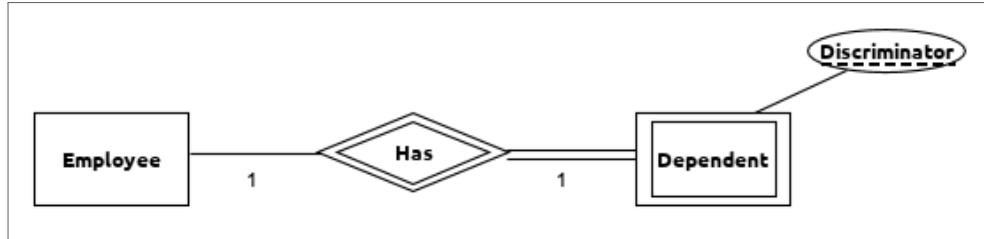


From: The author

R21 states that weak entity types must not have discriminator attributes if the identifier relationship type has 1:1 cardinality. According to Nascimento (2015), in the presence of a discriminator attribute, the weak entity type is uniquely identified by the values of the discriminator attribute and the strong entity type identifier attributes. Identifier relationships with 1:1 cardinality involving a weak entity type with a discriminator attribute, makes it possible for a strong entity type instance to uniquely identify more than one

weak entity type instance, which contradicts the cardinality constraint imposed on the relationship. Figure 36 shows an example of an invalid model.

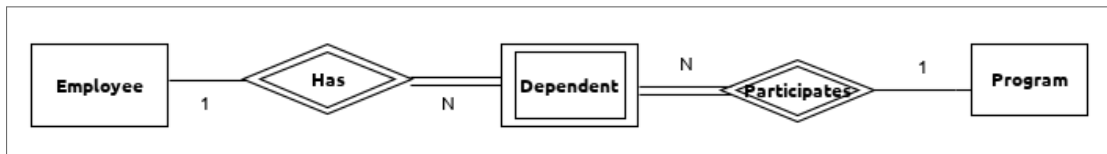
Figure 36 – Invalid model according to R21



From: The author

R22 defines that if a weak entity type participates in more than one identifier relationship, they all must have "1:N" cardinality. According to Nascimento (2015), in such cases, it is not possible to guarantee "1:1" cardinalities. Figure 37 shows an example of valid model according to R22.

Figure 37 – Invalid model according to R22



From: The author

R23 states that associative entities can not contain attributes (Figure 38). As mentioned before, the associative entity is a construct created only for making it possible to link two relationship types, and therefore should not have attributes.

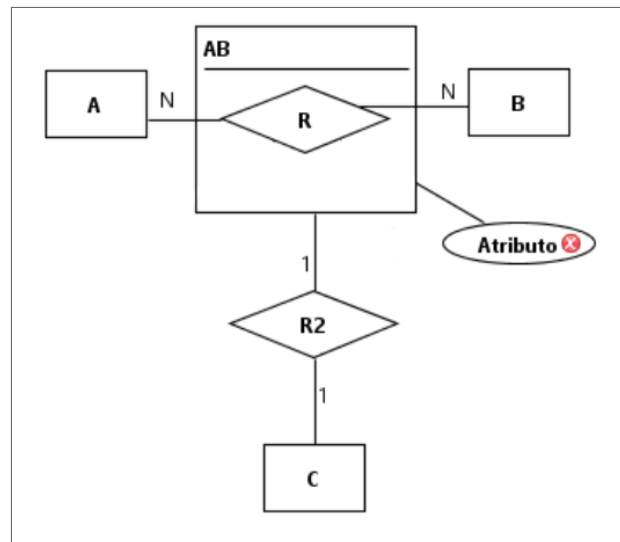
R24 defines that subclass entity types can not have identifier attributes (Figure 39). Since the inheritance relationship implies that the subclasses must inherit the superclasses attributes, it can not have an identifier attribute itself.

R25 asserts that subclass entity types can not have discriminator attributes (Figure 40). As mentioned before, discriminator attributes act as a partial identifier. Since sub-entities are uniquely identified by its attribute values, a partial identifier is not necessary.

R26 states that relationships types can not have identifier attributes. As discussed in chapter 2, only entity types can have identifier attributes. Figure 41 shows an example of an invalid model, according to R26.

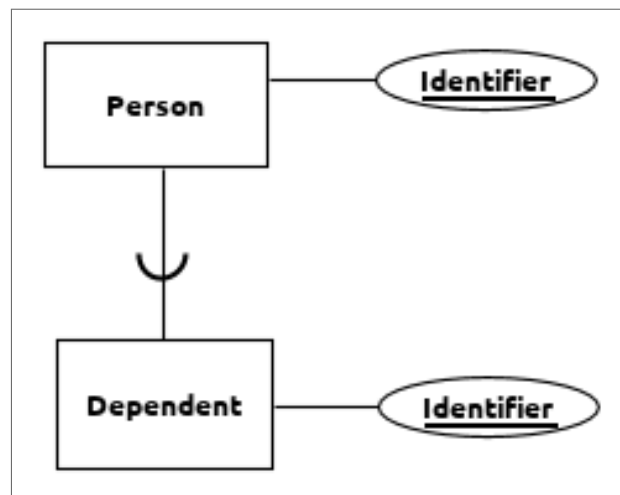
R27 states that only unary and binary relationship types with cardinality "N:N" may have discriminator attributes. An attribute should be added to a relationship when the business rule requires saving features that are directly related to the relationship between entities and not to the entities themselves. In N:N relationship types, this attribute may act as a partial identifier, i.e., as a discriminator attribute. On the remaining cardinalities,

Figure 38 – Invalid model according to R23



From: The author

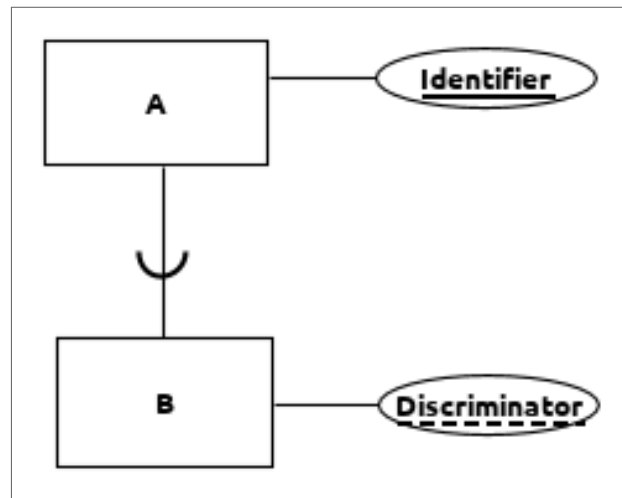
Figure 39 – Invalid model according to R24



From: The author

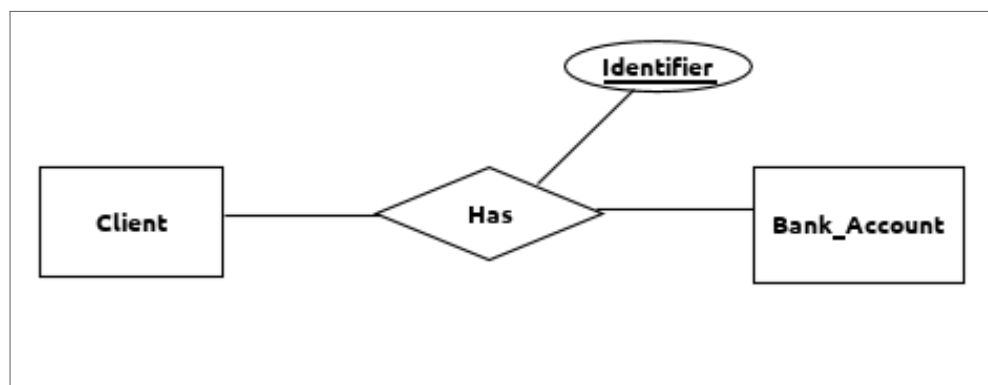
those attributes could be moved to the entity types with no damage to the model semantics (ELMASRI; NAVATHE, 2010). Figure 42 shows examples of invalid models according to R27.

Figure 40 – Invalid model according to R25



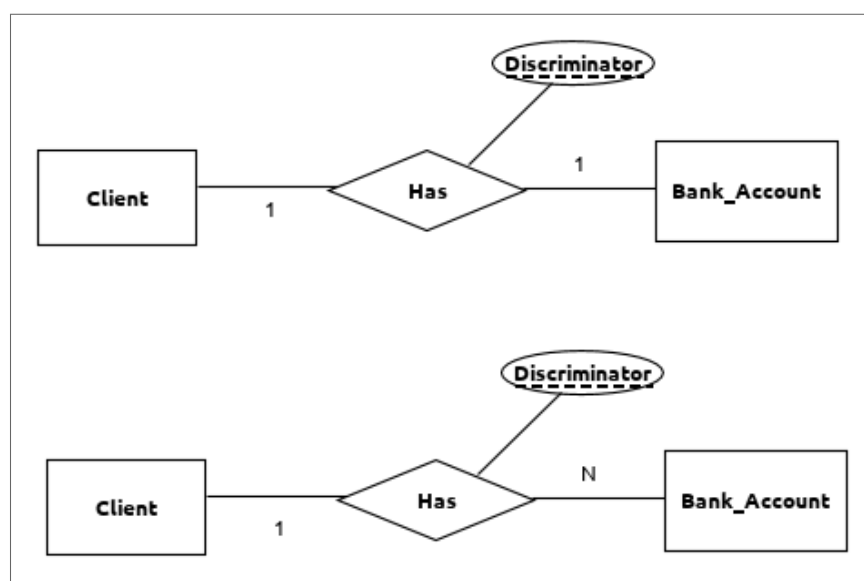
From: The author

Figure 41 – Invalid model according to R26



From: The author

Figure 42 – Invalid model according to R27



From: The author

3.2 FINAL REMARKS

In the first phase of a database project, the designer creates a conceptual model that is used as the basis for the next phases. Thus, the validation of these models is a critical task, as errors present in the model can be passed to the other phases, negatively affecting the final result. The sooner errors are detected, the lower the costs to fix them (WESTLAND, 2002). When validating conceptual models, it is necessary to take into account syntactic and semantic errors according to the modeling language used. It is also important to note that a model may be syntactically valid but semantically invalid. In this chapter, we present the catalog of validity rules organized by Nascimento (2015) and composed by semantic validity rules defined by the author, as well as rules found in the work of Dullea, Song and Lamprou (2003) and Calvanese and Lenzerini (1994).

Although the catalog has limitations, e.g., it does not cover the category constructor nor relationship types with degrees above 4, it has good coverage of the EER language constructors. Hence, it was used as a basis for the semantic validation of EER models proposed in this paper.

In the next chapter, we present the strategies used to represent both the syntactic validity rules based on the EER language abstract syntax and the semantic validity rules presented in this chapter.

4 EER KNOWLEDGE BASE

4.1 INTRODUCTION

As previously discussed, when validating a database conceptual model, we need to consider the syntax and semantics of the modeling language used. Hence, in this work, we proposed the validation of database conceptual models by automatically identifying syntactic and semantic inconsistencies in EER models using DL reasoners.

In order to achieve our goal, we first formalized the EER abstract syntax to a TBox. After that, we formalized the semantic validity rules from the catalog organized by Nascimento (2015) and presented in chapter 3. Although we tried to describe most of the rules by using axioms, we used Semantic Web Rule Language (SWRL) rules in the cases in which DL expressivity was not sufficient (e.g., cases involving cyclic paths).

We built our Knowledge Base with the support of Protégé (MUSEN, 2015), through which it was possible to create concepts (classes in OWL DL), roles (object properties in OWL DL), and some individuals. The use of the tool is justified by the intuitive graphical user interface, which allows visualizing the hierarchy between concepts and simplifies the creation of axioms. After creating the ontology in OWL DL, we manually converted it to an *ALCROIQ* KB.

Based on the above, this chapter presents our EER language Knowledge Base, describing the formalization of the EER abstract syntax in *ALCROIQ* and the strategies used to add the semantic validity rules. It also describes the procedures conducted to verify our KB.

4.2 ABSTRACT SYNTAX FORMALIZATION

Since the EER language does not have a standard abstract syntax, in this work, we considered the EERMM syntax (FIDALGO et al., 2013), which covers all EER constructs.

Figure 43 shows a representation of the EERMM by using the Ecore language, a UML dialect used to represent models in the Eclipse Modeling Framework (GRONBACK, 2017).

According to Figure 43, the EERMM includes the root concept "Schema", which represents the design area of the conceptual database diagram, and can be composed of several instances of Node and Link classes. The cardinality 0..* allows empty diagrams.

The Node class is a generalization of the following classes: Inheritance (which relates to the inheritance construct), Category (which corresponds to the category construct), and Element. The Element class is a generalization of the classes EntityType, Relationship (which refers to the relationship type concept), and Attribute (which corresponds to the attribute concept). Finally, AssociativeEntity and Entity are specializations of Enti-

tyType, that corresponds to the concepts of associative entity and entity type in the EER language, respectively.

The Link class has five specializations. They are: GeneralizationLink (specialized in InheritanceGL and CategoryGL), SpecializationLink (specialized in InheritanceSL and CategorySL), DirectInheritanceLink, RelationshipLink, and AttributeLink. Figure 44 shows all EERMM classes and their corresponding EER notations. Figure 45 shows an EER model annotated with its corresponding EERMM classes.

During our formalization process, we first focused on the following EERMM classes: "Inheritance", "DirectInheritanceLink", "GeneralizationLink" (and its subclasses) and "SpecializationLink" (and its subclasses). Since they are all related to the generalization/specialization relationship between entity types, we summarized them as the role "isSpecializationOf" and its inverse "isGeneralizationOf". Those roles proved to be adequate for model validation since the semantic validity rules considered in this work refer to the general concept of inheritance, as discussed in chapter 3. We omitted the "Category" class, since there is no semantic validity rules related to the construct in the catalog.

After that, we centered on the RelationshipLink class. To represent the class, we created a new role called "relationshipLink". We chose this approach based on the fact that the EER constructor represented by the RelationshipLink class behaves as a binary relationship involving an entity type and a relationship type. Hence, we stated the "relationshipLink" with the domain composed of instances of EntityType and the range composed of instances of Relationship. We also added its inverse role, "iRelationshipLink", with the domain composed of instances of "Relationship" and the range composed of instances of "EntityType". It is relevant to mention that we did not explicitly set the domain and range in Protégé in order to avoid unexpected side effects (HORRIDGE, 2011); instead, we used closing axioms. An example of a closing axiom would be: $EntityType \sqsubseteq \forall relationshipLink. Relationship$.

To represent the "AttributeLink" class, we created the relation "hasAttribute" and its inverse "isAttributeOf", following the same approach taken to represent the RelationshipLink class. In the end, we omitted the representation of the "Link" class itself, in order to avoid an unnecessary role hierarchy.

To represent the association between the "AssociativeEntity" class and the "Relationship", we created the "hasPart" role and its inverse "isPartOf".

Tables 5 and 6 show the RBox and the TBox, respectively, without the semantic validity rules.

Table 5 – EERMM Partial RBox

Roles
$isSpecializationOf \equiv isGeneralizationOf^{-}$

$relationshipLink \equiv iRelationshipLink^-$
$hasAttribute \equiv isAttributeOf^-$
$hasPart \equiv isPartOf^-$

From: The author

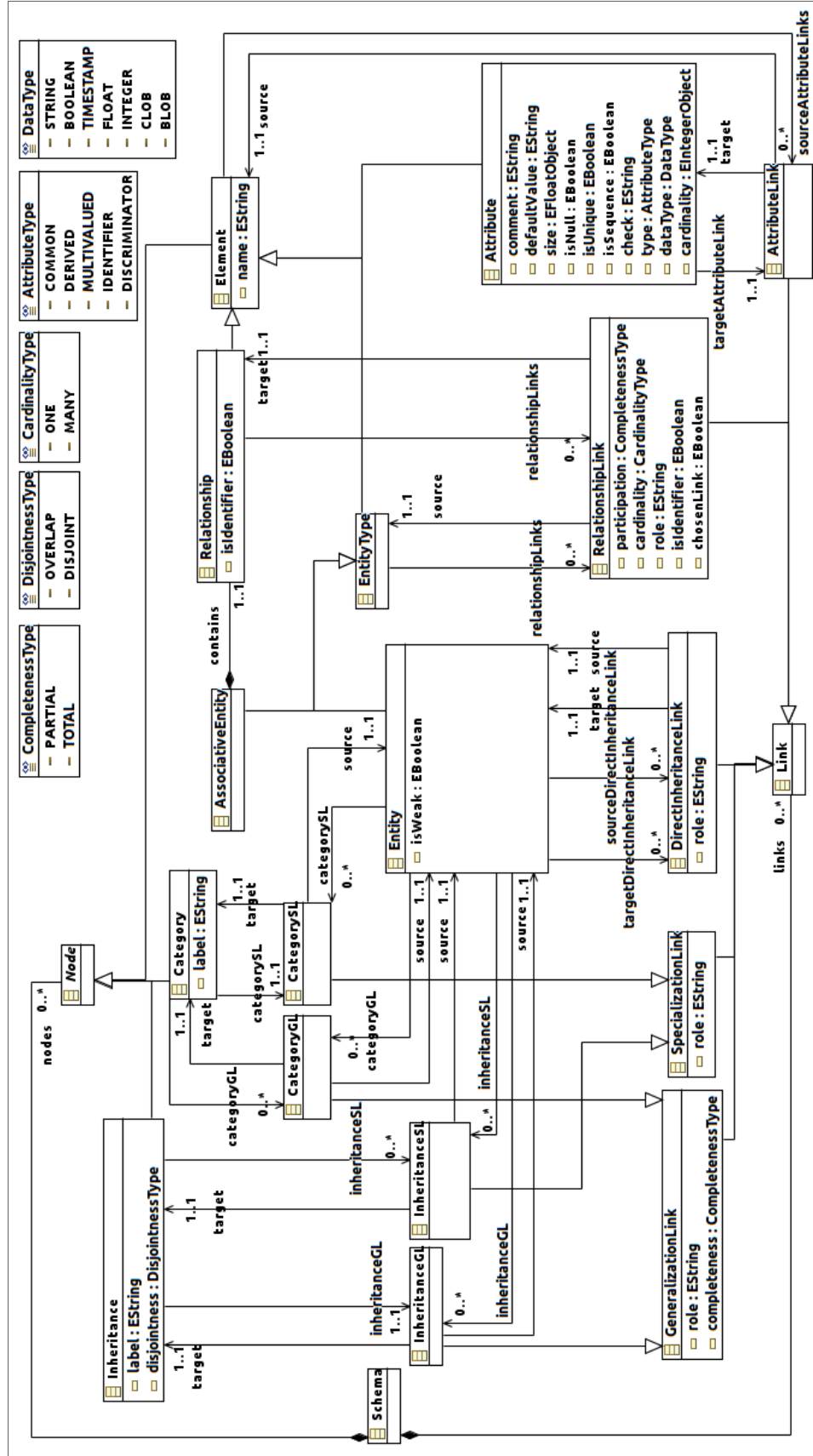
Table 6 – EERMM Partial TBox

Concept	Description
Schema	$\sqsubseteq \neg \exists isSpecializationOf. \top$ $\sqcap \neg \exists isGeneralizationOf. \top$ $\sqcap \neg \exists relationshipLink. \top$ $\sqcap \neg \exists iRelationshipLink. \top$ $\sqcap \neg \exists hasAttribute. \top$ $\sqcap \neg \exists isAttributeOf. \top$ $\sqcap \neg \exists isPartOf. \top$ $\sqcap \forall hasPart. Node$
Node	$\sqsubseteq \exists isPartOf. Schema$
Element	$\sqsubseteq Node$ $\sqcap \forall hasAttribute. Attribute$
Attribute	$\sqsubseteq Element$ $\sqcap \neg \exists isSpecializationOf. \top$ $\sqcap \neg \exists isGeneralizationOf. \top$ $\sqcap \neg \exists relationshipLink. \top$ $\sqcap \neg \exists iRelationshipLink. \top$ $\sqcap \forall isAttributeOf. Element$ $\sqcap \neg \exists hasPart. \top$ $\sqcap \forall isPartOf. Schema$
EntityType	$\sqsubseteq Element$ $\sqcap \forall relationshipLink. Relationship$ $\sqcap \neg \exists iRelationshipLink. \top$ $\sqcap \neg \exists isAttributeOf. \top$ $\sqcap \forall isPartOf. Schema$

AssociativeEntity	$\sqsubseteq \text{EntityType}$ $\Box \neg \exists \text{isSpecializationOf}.\top$ $\Box \neg \exists \text{isGeneralizationOf}.\top$ $\Box \forall \text{hasPart.Relationship}$ $\Box = 1 \text{hasPart.Relationship}$
Entity	$\sqsubseteq \text{EntityType}$ $\Box \forall \text{isSpecializationOf.Entity}$ $\Box \forall \text{isGeneralizationOf.Entity}$ $\Box \neg \exists \text{hasPart}.\top$
Relationship	$\sqsubseteq \text{Element}$ $\Box \neg \exists \text{isSpecializationOf}.\top$ $\Box \neg \exists \text{isGeneralizationOf}.\top$ $\Box \neg \exists \text{relationshipLink}.\top$ $\Box \forall i \text{RelationshipLink.EntityType}$ $\Box \neg \exists \text{isAttributeOf}.\top$ $\Box \forall \text{isPartOf}.(AssociativeEntity \sqcup Schema)$ $\Box \neg \exists \text{hasPart}.\top$

From: The author

Figure 43 – EERMM Metamodel



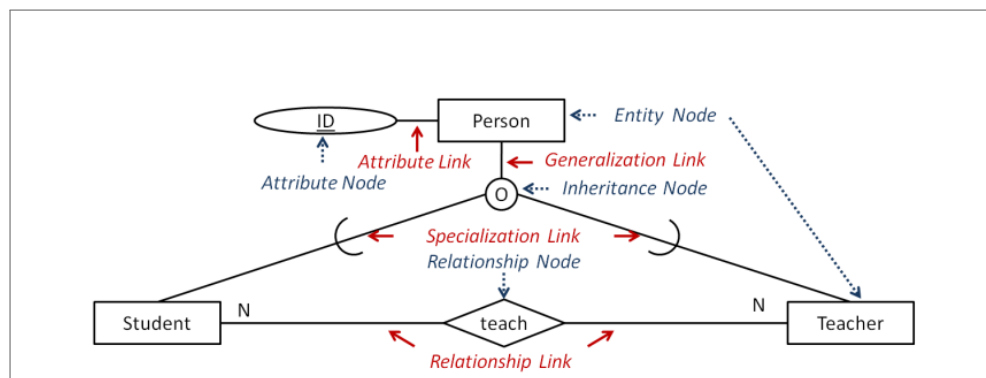
From: Fidalgo et al. (2013)

Figure 44 – EERMM concepts

	EER CONSTRUCTORS	EER META-ENTITIES
Nodes		ENTITY
		RELATIONSHIP
		ATTRIBUTE
		ASSOCIATIVE_ENTITY
		INHERITANCE
		CATEGORY
Links		RELATIONSHIP_LINK
		ATTRIBUTE_LINK
		INHERITANCE_SL
		CATEGORY_SL
		INHERITANCE_GL
		CATEGORY_GL
		DIRECT_INHERITANCE_LINK

From: Fidalgo et al. (2013)

Figure 45 – EERMM concepts



From: Fidalgo et al. (2012)

4.3 SEMANTIC VALIDITY RULES FORMALIZATION

After creating the KB based on the EERMM, we started adding the semantic validity rules described in chapter 3.

First, we focused on rules R01 to R04, related to cardinality and participation constraints in unary relationship types. As shown in Figure 43, the EERMM represents such constraints as the "cardinality" and "participation" attributes, which are part of the "RelationshipLink" class. In order to represent them in the KB, we created the following roles as children of "relationshipLink": "relationshipLinkTotal" (refers to total participation) and "relationshipLinkPartial" (relates to partial participation). We also included the subroles "relationshipLinkTotalOne" (refers to total participation and one cardinality), "relationshipLinkTotalMany" (relates to total participation and many cardinality), "relationshipLinkPartialOne" (regards partial participation and one cardinality), and "relationshipLinkPartialMany" (relates to partial participation and many cardinality). Finally, we added their inverse roles as children of "iRelationshipLink", following the same naming pattern and adding the prefix "i". Table 7 shows the hierarchy.

Table 7 – *relation* and *iRelation* hierarchies

Roles
$relationshipLinkPartial \sqsubseteq relationshipLink$
$relationshipLinkPartialOne \sqsubseteq relationshipLinkPartial$
$relationshipLinkPartialMany \sqsubseteq relationshipLinkPartial$
$relationshipLinkTotal \sqsubseteq relationshipLink$
$relationshipLinkTotalOne \sqsubseteq relationshipLinkTotal$
$relationshipLinkTotalMany \sqsubseteq relationshipLinkTotal$
$iRelationshipLinkPartial \sqsubseteq iRelationshipLink$
$iRelationshipLinkPartialOne \sqsubseteq iRelationshipLinkPartial$
$iRelationshipLinkPartialMany \sqsubseteq iRelationshipLinkPartial$
$iRelationshipLinkTotal \sqsubseteq iRelationshipLink$
$iRelationshipLinkTotalOne \sqsubseteq iRelationshipLinkTotal$
$iRelationshipLinkTotalMany \sqsubseteq iRelationshipLinkTotal$

From: The author

After adding cardinality and participation constraints to the KB, we started to formalize the rules R01 to R04.

R01 asserted that "all 1:1 unary relationship types with total-partial or partial-total participation are structurally invalid." To formalize this rule, we stated "iRelationshipLinkTotalOne" and "iRelationshipLinkPartialOne" as disjoint roles.

R02 stated that "all 1 : M or M : 1 unary relationship types with total-total participation are structurally invalid." To add this rule, we stated iRelationshipLinkTotalOne

and `iRelationshipLinkTotalMany` as disjoint roles.

R03 asserted that "all 1 : M or M : 1 unary relationship types with total participation on the ‘one’ side and partial participation on the ‘many’ side are structurally invalid." To add this rule, we stated `iRelationshipLinkTotalOne` and `iRelationshipLinkPartialMany` as disjoint roles. At this point, by considering R01, as well as the role hierarchy, we simplified the rules by stating `iRelationshipLinkTotalOne` and `iRelationshipLinkPartial` as disjoint roles.

R04 stated that identifier relationship types must not be unary. The rule depends on the relationship degree, which can not be identified by merely considering the number of entities explicitly related to a relationship, because of the DL open world assumption. To solve this problem, we added the "RelationshipDegree", "UnaryRelationship", "BinaryRelationship", and "TernaryRelationship" concepts to the KB. Regarding the relationship type, we added the concepts "RegularRelationship" and "IdentifierRelationship" as children of "Relationship". We also stated that identifier relationships must not be unary, as shown in Table 8. When creating the ABox we must assign the degree and the type of each relationship.

Table 8 shows a summary of the formalization strategies. We have omitted the closing axioms for simplicity.

Table 8 – Summary of strategies: unary relationship types

Roles	Formalization Strategy
R01, R03	$\text{Disjoint}(iRelationshipLinkTotalOne, iRelationshipLinkPartial)$
R02	$\text{Disjoint}(iRelationshipLinkTotalOne, iRelationshipLinkTotalMany)$
R04	RelationshipDegree UnaryRelationship \sqsubseteq <i>RelationshipDegree</i> $\Box \neg (BinaryRelationship \sqcup TernaryRelationship)$ BinaryRelationship \sqsubseteq <i>RelationshipDegree</i> $\Box \neg (UnaryRelationship \sqcup TernaryRelationship)$ TernaryRelationship \sqsubseteq <i>RelationshipDegree</i> $\Box \neg (UnaryRelationship \sqcup BinaryRelationship)$ RegularRelationship \sqsubseteq <i>Relationship</i> $\Box \neg IdentifierRelationship$ IdentifierRelationship \sqsubseteq <i>Relationship</i> $\Box \neg RegularRelationship$ $\Box \neg UnaryRelationship$

From: The author

After formalizing the rules related to unary relationships, we started working on the rules R05 to R10, related to binary relationship types.

R05 asserts that "cyclic paths containing no opposing and no self-adjusting binary relationship types are structurally invalid." To identify those inconsistencies, Dullea, Song and Lamprou (2003) analyzed the restrictions on the number of instances between the entity types involved in a binary relationship and provided a table (Table 9) with the results of the analysis. Since R05 regards the relationship between two instances of the "Entity" class, and the interaction of cardinality, participation, and cyclic paths, we had to apply a new strategy.

Table 9 – Structural restrictions of a binary relationship.

Case	Cardinality	Participation	Restrictions
1	1:1	Total-Total	$ E_1 = E_2 $
2	1:1	Total-Partial	$ E_1 < E_2 $
3	1-1	Partial-Total	$ E_1 > E_2 $
4	1:1	Partial-Partial	Self-adjusting
5	N:1	Total-Total	$ E_1 > E_2 $
6	N:1	Total-Partial	Self-adjusting
7	N:1	Partial-Total	$ E_1 > E_2 $
8	N:1	Partial-Partial	Self-adjusting
9	1:N	Total-Total	$ E_1 < E_2 $
10	1:N	Total-Partial	$ E_1 < E_2 $
11	1:N	Partial-Total	Self-adjusting
12	1:N	Partial-Partial	Self-adjusting
13	N:N	Total-Total	Self-adjusting
14	N:N	Total-Partial	Self-adjusting
15	N:N	Partial-Total	Self-adjusting
16	N:N	Partial-Partial	Self-adjusting

Adapted from: Dullea, Song and Lamprou (2003)

At this point, it is important to remember that the EERMM Entity class corresponds to the concept of entity type in the EER language, i.e., to the schemas or classes responsible for describing the structure of a set of entity objects (or instances) (ELMASRI; NAVATHE, 2010). Based on that, we added the role "isSmallerThan", to represent the cases when the number of instances of an Entity E_1 is smaller than the number of instances of an Entity E_2 , and the role "isGreaterThan", to represent the opposing cases. Since those roles are inverse roles (see Table 9), we decided to remove the "isGreaterThan" one in order to keep the simplicity and to avoid redundancy. We also stated that the number of instances of an Entity might not be smaller than its number of instances by adding the axiom $Entity \sqsubseteq \neg \exists isSmallerThan.Self$.

To identify the cases that imply in inequalities, we created the SWRL rules shown in Figures 46, 47, and 48 which are respectively related to the cases 2, 9, and 10 presented in

Table 9. We also added the SWRL rule in Figure 49 in order to assure the *isSmallerThan* role transitivity, which was necessary because, in OWL DL, transitive properties cannot be asymmetric in order to ensure decidability.

Figure 46 – SWRL Rule: Case 2

SWRL Rule: Case 2 (1:1, partial-total)

```
Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
BinaryRelationship(?r) ^
iRelationshipLinkTotalOne(?r, ?x) ^
iRelationshipLinkPartialOne(?r, ?y)
-> isSmallerThan(?x, ?y)
```

From: The author

Figure 47 – SWRL Rule: Case 9

SWRL Rule: Case 9 (1:N, total-total)

```
Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
BinaryRelationship(?r) ^
iRelationshipLinkTotalOne(?r, ?x) ^
iRelationshipLinkTotalMany(?r, ?y)
-> isSmallerThan(?x, ?y)
```

From: The author

Figure 48 – SWRL Rule: Case 10

SWRL Rule: Case 10 (1:N, total-partial)

```
Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
BinaryRelationship(?r) ^
iRelationshipLinkTotalOne(?r, ?x) ^
iRelationshipLinkPartialMany(?r, ?y)
-> isSmallerThan(?x, ?y)
```

From: The author

Figure 49 – SWRL Rule: *isSmallerThan* transitivity

SWRL Rule: *isSmallerThan* Transitivity

```
Entity(?x) ^ Entity(?y) ^ Entity(?z) ^
differentFrom(?x, ?y) ^ differentFrom(?y, ?z) ^
isSmallerThan(?x, ?y) ^ isSmallerThan(?y, ?z)
-> isSmallerThan(?x, ?z)
```

From: The author

After that, we included the *isEqualTo* role to represent the case when the number of instances of an *Entity* E_1 is equal to the number of instances of an *Entity* E_2 . The *isEqualTo* role is necessary because, although the presence of an equal number of instances has a neutral effect in a cyclic path, it closes the path making it possible to validate the inequalities. Based on the case 1 in Table 9, we added the SWRL rule shown in Figure 50. We also added the axiom $Entity \sqsubseteq \exists isEqualTo.Self$ in order to state that the number of instances of an Entity is equal to itself. We assured the role transitivity with the SWRL rule shown in Figure 51.

Figure 50 – SWRL Rule: Case 1

SWRL Rule: Case 1 (1:1, total-total)

```
Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
BinaryRelationship(?r) ^
iRelationshipLinkTotalOne(?r, ?x) ^
iRelationshipLinkTotalOne(?r, ?y)
-> isEqualTo(?x, ?y)
```

From: The author

Figure 51 – SWRL Rule: isEqualsTo transitivity

SWRL Rule: isEqualTo Transitivity

```
Entity(?x) ^ Entity(?y) ^ Entity(?z) ^
differentFrom(?x, ?y) ^ differentFrom(?x, ?z) ^
differentFrom(?y, ?z) ^ isEqualTo(?x, ?y) ^
isEqualTo(?y, ?z) -> isEqualTo(?x, ?z)
```

From: The author

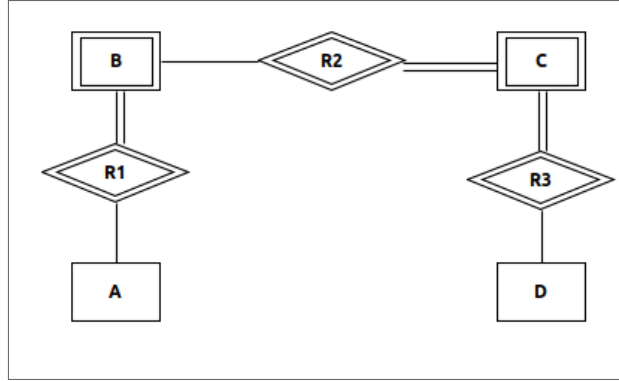
We did not include the self-adjusting cases because their presence automatically implies in a valid model.

R06 states that associative entities must not have total participation in relationships. We added the following axiom in order to ensure this constraint: $AssociativeEntity \sqsubseteq \neg \exists relationshipLinkTotal.Relationship$.

R07 defines that no strong entity type link in an identifier relationship type can have cardinality "N". As discussed in chapter 3, we classify an entity type as weak when it is uniquely identified through its relationship with another entity type, which we call a strong entity type. Although the EER language has different notations for representing strong and weak entity types, in an EER model, we can represent the same entity as weak and strong, depending on the relationship that we are analyzing. In the example shown in Figure 52, the entity type C is weak in the relationship with R_3 but might be strong in the relationship with R_2 , although its notation in the model always corresponds to that of a weak entity type (double rectangle). In the example, we say that entity type C "might

be strong" because it is not explicit in the model which of the entities participating in the relationship R_2 is, in fact, the weak entity, since they have the same notation.

Figure 52 – Weak and Strong Entities



From: Moe (2004)

The EERMM handles this problem through the "isIdentifier" attribute present in the "RelationshipLink" class. Then, the model creator must specify which entity type should be considered weak by assigning the value "true" to the attribute of the corresponding "RelationshipLink". Based on that, we added the roles "relationshipLinkIdentifier", representing the relationship between the weak entity type and the identifier relationship type, and "relationshipLinkNonIdentifier", representing the relationship between the strong entity type and the identifier relationship type. We also added the inverse roles "iRelationshipLinkIdentifier" and "iRelationshipLinkNonIdentifier", respectively. Finally, we marked "iRelationshipLinkIdentifier" and "iRelationshipLinkNonIdentifier" as disjoint, since the relation between a "Relationship" and an "Entity" should not be identifier and non-identifier at the same time. Then, to add R07, we marked "iRelationshipLinkNonIdentifier" as disjoint with "iRelationshipLinkTotalMany" and "iRelationshipLinkPartialMany".

R08 defines that an identifier relationship type can not be doubly identified. In order to formalize this rule, we considered the existence dependency concept discussed in chapter 3, by adding the role "dependsOn". We also included the following axiom to state that an entity type can not be existence dependent on itself: $Entity \sqsubseteq \neg \exists dependsOn.Self$. Next, we included the SWRL rule described in Figure 53 in order to identify when an "Entity" is existentially dependent of another "Entity". Finally, we added the SWRL rule shown in Figure 54, that makes "dependsOn" transitive.

R09 states that identifier relationship types are not allowed to form a cyclic path. As discussed in chapter 3, this rule considers the concept of existence dependency and is also caught by the "dependsOn" role added for R08.

R10 defines that the identifying links between a weak entity type and an identifier relationship type must have "total" participation. In order to assure the constraint, we marked the "iRelationshipLinkIdentifier" role as disjoint with "iRelationshipLinkPartial".

Figure 53 – SWRL Rule: dependsOn

SWRL Rule: dependsOn

```

Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
IdentifierRelationship(?r) ^
iRelationshipLinkIdentifier(?r, ?x) ^
iRelationshipLink(?r, ?y)
-> dependsOn(?x, ?y)

```

From: The author

Figure 54 – SWRL Rule: dependsOn transitivity

SWRL Rule: dependsOn Transitivity

```

Entity(?x) ^ Entity(?y) ^ Entity(?z) ^
differentFrom(?x, ?y) ^ differentFrom(?y, ?z) ^
dependsOn(?x, ?y) ^ dependsOn(?y, ?z)
-> dependsOn(?x, ?z)

```

From: The author

Table 10 shows a summary of the formalization strategies.

Table 10 – Summary of strategies: binary relationship types

Roles	Formalization Strategy
R05	$Entity \sqsubseteq \neg \exists isSmallerThan.Self$ $Entity \sqsubseteq \exists isEqualsTo.Self$ SWRL Rule: Case 1 SWRL Rule: Case 2 SWRL Rule: Case 9 SWRL Rule: Case 10 SWRL Rule: isSmallerThan transitivity SWRL Rule: isEqualTo transitivity
R06	$AssociativeEntity \sqsubseteq \neg \exists relationshipLinkTotal.Relationship$
R07	$relationshipLinkIdentifier \equiv iRelationshipLinkIdentifier^-$ $relationshipLinkNonIdentifier \equiv iRelationshipLinkNonIdentifier^-$ $Disjoint(iRelationshipLinkIdentifier, iRelationshipLinkNonIdentifier)$ $Disjoint(iRelationshipLinkNonIdentifier, iRelationshipLinkTotalMany)$ $Disjoint(iRelationshipLinkNonIdentifier, iRelationshipLinkPartialMany)$
R0[8-9]	$Entity \sqsubseteq \neg \exists dependsOn.Self$ SWRL Rule: dependsOn SWRL Rule: dependsOn transitivity
R10	$Disjoint(iRelationshipLinkIdentifier, iRelationshipLinkPartial)$

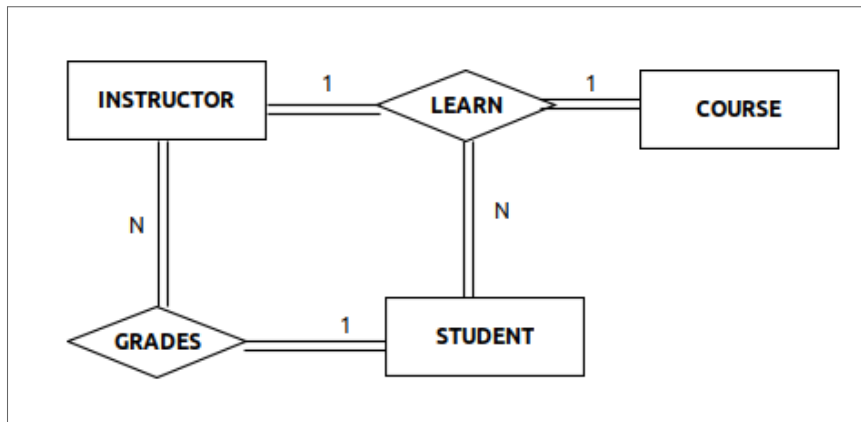
From: The author

Next, we added the rules R11 to R13 related to ternary relationship types.

R11 states that cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the cardinality constraints of the binary relationship type are less than the cardinality constraints of the ternary one (Figure 55). As discussed in chapter 3, since the EER language has no constructors to indicate when a binary relationship type constrains a ternary relationship type, structural errors according to R11 must be checked against business rules. Based on that, we decided to add R11 to our KB as a warning to the database designer, considering that all binary relationship types linked to ternary relationship types are related to the same business rules.

R11 is directly related to the relationship degree and the relationship between "Relationship" instances. Since, in the EER language, there are no explicit relations between relationship types, we added the roles "linkedRelationships", to represent the link between a binary and a ternary relationship, and "smallerCardinality", in order to identify when a binary relationship type has cardinality constraints smaller than the ternary relationship type. The SWRL rules presented in Figures 56 and 57 are responsible to identify the cases mentioned. Finally, we also marked the roles as disjoint since they must not occur at the same time between the same individuals.

Figure 55 – R11 Example



From: Dullea, Song and Lamprou (2003)

Figure 56 – SWRL Rule: linkedRelationships

SWRL Rule: linkedRelationships

```
Entity(?x) ^ Entity(?y) ^ differentFrom(?x, ?y) ^
BinaryRelationship(?r1) ^
TernaryRelationship(?r2) ^
iRelationshipLink(?r1, ?x) ^
iRelationshipLink(?r1, ?y) ^
iRelationshipLink(?r2, ?x) ^
iRelationshipLink(?r2, ?y)
-> linkedRelationships(?r1, ?r2)
```

From: The author

Figure 57 – SWRL Rule: smallerCardinality

SWRL Rules: smallerCardinality

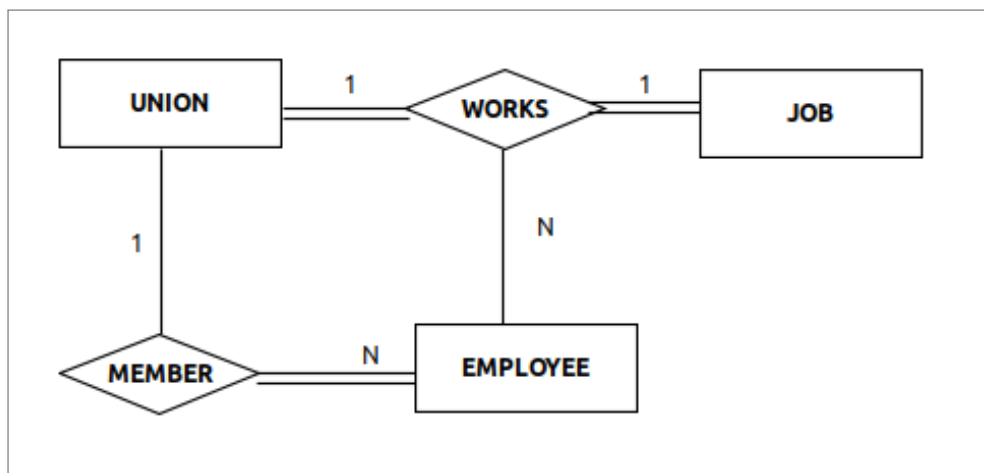
```
Entity(?x) ^ BinaryRelationship(?r1) ^
TernaryRelationship(?r2) ^
iRelationshipLinkPartialOne(?r1, ?x) ^
iRelationshipLinkPartialMany(?r2, ?x)
-> smallerCardinality(?r1, ?r2)
```

```
Entity(?x) ^ BinaryRelationship(?r1) ^
TernaryRelationship(?r2) ^
iRelationshipLinkTotalOne(?r1, ?x) ^
iRelationshipLinkTotalMany(?r2, ?x)
-> smallerCardinality(?r1, ?r2)
```

From: The author

R12 defined that cyclic paths that contain ternary relationship types constrained by a binary relationship type are invalid if the participation constraints of the binary relationship type are less than the participation constraints of the ternary one (Figure 58). Following the same strategy as in R11, we also added R12 to our KB. Since this rule also depends on the binary relationship between instances of Relationship, we considered the previously created role "linkedRelationships", and created a new one, called "differentParticipation", and marked them as disjoint. We also added the SWRL rules presented in Figure 59 in order to catch the cases mentioned.

Figure 58 – R12 Example

**From:** Dullea, Song and Lamprou (2003)

R13 asserted that a relationship type must not be ternary and identifier at the same time. Hence, we added the following axiom to ensure this constraint:

$$IdentifierRelationship \sqsubseteq \neg TernaryRelationship.$$

Figure 59 – SWRL Rule: differentParticipation

SWRL Rules: differentParticipation

```
Entity(?x) ^ BinaryRelationship(?r1) ^
TernaryRelationship(?r2) ^
iRelationshipLinkPartial(?r1, ?x) ^
iRelationshipLinkTotal(?r2, ?x)
-> differentParticipation(?r1, ?r2)
```

```
Entity(?x) ^ BinaryRelationship(?r1) ^
TernaryRelationship(?r2) ^
iRelationshipLinkTotal(?r1, ?x) ^
iRelationshipLinkPartial(?r2, ?x)
-> differentParticipation(?r1, ?r2)
```

From: The author

Table 11 shows a summary of the strategies.

Table 11 – Summary of strategies: ternary relationship types

Roles	Formalization Strategy
R11	<i>Disjoint(linkedRelationships, smallerCardinality)</i> SWRL Rules: linkedRelationships, smallerCardinality
R12	<i>Disjoint(linkedRelationships, differentParticipation)</i> SWRL Rules: differentParticipation
R13	<i>IdentifierRelationship $\sqsubseteq \neg$ TernaryRelationship</i>

From: The author

After, we added inheritance constraints corresponding to the rules R14 to R18.

R14 defines that a model must not have inheritances forming a cyclic path. In order to catch this case, we added the following axiom to ensure that an "Entity" can not be a child of itself: $Entity \sqsubseteq \neg \exists isSpecializationOf.Self$. We also added the SWRL rule shown in Figure 60 to assure the "isSpecializationOf" transitivity.

R15 states that an entity type must not be weak and a subclass in the same model. To assure this constraint, we added the axiom shown in Table 12.

R16 defines that superclass entity types must not be weak entities of its subclasses. As discussed in chapter 3, the relationship between the weak entity type and the strong entity type comprises the concept of existence dependency, which is caught by the "dependsOn" role added for R08 and R09. To add R16, we marked the roles "dependsOn" and "isGeneralizationOf" as disjoint.

Figure 60 – SWRL Rule: isSpecializationOf transitivity

SWRL Rules: isSpecializationOf Transitivity

```

Entity(?x) ^ Entity(?y) ^ Entity(?z) ^
differentFrom(?x, ?y) ^ differentFrom(?y, ?z) ^
isSpecializationOf(?x, ?y) ^
isSpecializationOf(?y, ?z)
-> isSpecializationOf(?x, ?z)

```

From: The author

R17 states that models that present relationship types that result in restrictions on the number of instances different from those imposed by the inheritances are structurally invalid. Since inheritance imposes that the number of instances of a superclass entity type must not be less than the number of instances of its subclass entity types, we added this restriction by making "isSmallerThan" and "isGeneralizationOf" disjoint.

Finally, R18 determines that associative entities must not participate in inheritances. The EERMM metamodel restricts it with the axioms presented in Table 6 and highlighted in Table 12.

Table 12 – Summary of strategies: inheritance

Roles	Formalization Strategy
R14	$Entity \sqsubseteq \neg \exists isSpecializationOf.Self$ SWRL Rule: isSpecializationOf transitivity
R15	$Entity \sqsubseteq \neg ((\exists isSpecializationOf.Entity) \sqcap (\exists relationshipLinkIdentifier.IdentifierRelationship))$
R16	$Disjoint(dependsOn, isGeneralizationOf)$
R17	$Disjoint(isSmallerThan, isGeneralizationOf)$
R18	$AssociativeEntity \sqsubseteq \neg \exists isSpecializationOf.\top$ $\sqcap \neg \exists isGeneralizationOf.\top$

From: The author

Finally, we added the rules R19 to R27, related to attributes.

R19 states that regular entity types should not have a discriminator attribute. As discussed in chapter 3, a regular entity is an entity that is not identified by its relationship to another entity. Thus, the concept of regular entity comprises entities that do not participate in identifier relationships and those that act as strong entities in all the identifier relationships in which they participate. In both cases, entities are represented by the regular entity notation (a rectangle). In EERMM, the regular entity notation is attributed to an entity when we assign the value "false" to the "isWeak" attribute of the "Entity" class. Likewise, the weak entity notation is attributed when we assign the value "true" to "isWeak". In order to identify those cases, we included the concepts "EntityNotation",

"RegularEntity" and "WeakEntity", representing the possible notations. We also added the concept "DiscriminatorAttribute" as a child of "Attribute". After that, we declared that instances of "RegularEntity" must not have a "DiscriminatorAttribute" as shown in Table 13.

R20 states that weak entity types must not have identifier attributes. Following the same strategy used in R19, we added the concept "IdentifierAttribute" as a child of "Attribute" and declared that instances of "WeakEntity" must not have a "IdentifierAttribute".

R21 states that Weak entity types must not have discriminator attributes if the identifier relationship type has 1:1 cardinality. Based on R07 and R10, we added the axiom presented in Table 13 to add this constraint.

R22 defines that if a weak entity type participates in more than one identifier relationship, they all must have "1:N" cardinality. Again based on rules R07 and R10, we added this constraint by including the axiom shown in Table 13.

The formalization strategies used for rules R23 to R27 are simple and straightforward, as presented in Table 13, and therefore, will not be discussed.

Table 13 – Summary of strategies: attribute type

Roles	Formalization Strategy
R19	EntityNotation RegularEntity \sqsubseteq <i>EntityNotation</i> \sqcap \neg <i>WeakEntity</i> WeakEntity \sqsubseteq <i>EntityNotation</i> \sqcap \neg <i>RegularEntity</i> DiscriminatorAttribute \sqsubseteq <i>Attribute</i> RegularEntity \sqsubseteq $\neg\exists$ <i>hasAttribute.DiscriminatorAttribute</i>
R20	IdentifierAttribute \sqsubseteq <i>Attribute</i> WeakEntity \sqsubseteq $\neg\exists$ <i>hasAttribute.IdentifierAttribute</i>
R21	Entity \sqsubseteq $\neg((\exists$ <i>hasAttribute.DiscriminatorAttribute</i>) $\sqcap(\exists$ <i>relationshipLinkIdentifier.IdentifierRelationship</i>) $\sqcap(\exists$ <i>relationshipLinkTotalOne.IdentifierRelationship</i>))
R22	Entity \sqsubseteq $\neg((\exists$ <i>relationshipLinkTotalOne.IdentifierRelationship</i>) $\sqcap(\geq 2$ <i>relationshipLinkIdentifier.IdentifierRelationship</i>)
R23	AssociativeEntity \sqsubseteq $\neg\exists$ <i>hasAttribute.</i> \top
R24	Entity \sqsubseteq $\neg((\exists$ <i>hasAttribute.IdentifierAttribute</i>) $\sqcap(\exists$ <i>isSpecializationOf.Entity</i>))
R25	Entity \sqsubseteq $\neg((\exists$ <i>hasAttribute.DiscriminatorAttribute</i>) $\sqcap(\exists$ <i>isSpecializationOf.Entity</i>))

R26	Relationship $\sqsubseteq \neg \exists hasAttribute.IdentifierAttribute$
R27	UnaryRelationship $\sqsubseteq \neg(((iRelationshipLinkPartialOne.Entity)$ $\sqcup (\exists iRelationshipLinkTotalOne.Entity))$ $\sqcap (\exists hasAttribute.DiscriminatorAttribute))$ BinaryRelationship $\sqsubseteq \neg(((iRelationshipLinkPartialOne.Entity)$ $\sqcup (\exists iRelationshipLinkTotalOne.Entity))$ $\sqcap (\exists hasAttribute.DiscriminatorAttribute))$

From: The author

Tables 14 and 15 show the RBox and the TBox, respectively, including the structural semantics constraints.

Table 14 – EERMM RBox

Roles
$isSpecializationOf \equiv isGeneralizationOf^-$
$relationshipLink \equiv iRelationshipLink^-$
$relationshipLinkPartial \sqsubseteq relationshipLink$
$relationshipLinkPartialOne \sqsubseteq relationshipLinkPartial$
$relationshipLinkPartialMany \sqsubseteq relationshipLinkPartial$
$relationshipLinkTotal \sqsubseteq relationshipLink$
$relationshipLinkTotalOne \sqsubseteq relationshipLinkTotal$
$relationshipLinkTotalMany \sqsubseteq relationshipLinkTotal$
$relationshipLinkPartial \equiv iRelationshipLinkPartial^-$
$relationshipLinkPartialOne \equiv iRelationshipLinkPartialOne^-$
$relationshipLinkPartialMany \equiv iRelationshipLinkPartialMany^-$
$relationshipLinkTotal \equiv iRelationshipLinkTotal^-$
$relationshipLinkTotalOne \equiv iRelationshipLinkTotalOne^-$
$relationshipLinkTotalMany \equiv iRelationshipLinkTotalMany^-$
$iRelationshipLinkPartial \sqsubseteq iRelationshipLink$
$iRelationshipLinkPartialOne \sqsubseteq iRelationshipLinkPartial$
$iRelationshipLinkPartialMany \sqsubseteq iRelationshipLinkPartial$
$iRelationshipLinkTotal \sqsubseteq iRelationshipLink$

$iRelationshipLinkTotalOne \sqsubseteq iRelationshipLinkTotal$
$iRelationshipLinkTotalMany \sqsubseteq iRelationshipLinkTotal$
$hasAttribute \equiv isAttributeOf^-$
$hasPart \equiv isPartOf^-$
$isSmallerThan$
$isEqualTo$
$linkedRelationships$
$smallerCardinality$
$differentParticipation$
$dependsOn$
$relationshipLinkIdentifier \equiv iRelationshipLinkIdentifier^-$
$relationshipLinkNonIdentifier \equiv iRelationshipLinkNonIdentifier^-$
$Disjoint(iRelationshipLinkNonIdentifier, iRelationshipLinkTotalMany)$
$Disjoint(iRelationshipLinkNonIdentifier, iRelationshipLinkPartialMany)$
$Disjoint(iRelationshipLinkIdentifier, iRelationshipLinkNonIdentifier)$
$Disjoint(iRelationshipLinkIdentifier, iRelationshipLinkPartial)$
$Disjoint(iRelationshipLinkTotalOne, iRelationshipLinkPartial)$
$Disjoint(iRelationshipLinkTotalOne, iRelationshipLinkTotalMany)$
$Disjoint(isSmallerThan, isEqualTo)$
$Disjoint(linkedRelationships, smallerCardinality)$
$Disjoint(linkedRelationships, differentParticipation)$
$Disjoint(dependsOn, isGeneralizationOf)$
$Disjoint(isSmallerThan, isGeneralizationOf)$

From: The author

Table 15 – EERMM TBox

Concept	Description
---------	-------------

Schema	$\begin{aligned} &\sqsubseteq \neg \exists isSpecializationOf.\top \\ &\sqcap \neg \exists isGeneralizationOf.\top \\ &\sqcap \neg \exists relation.\top \\ &\sqcap \neg \exists iRelation.\top \\ &\sqcap \neg \exists hasAttribute.\top \\ &\sqcap \neg \exists isAttributeOf.\top \\ &\sqcap \neg \exists isPartOf.\top \\ &\sqcap \neg \exists dependsOn.\top \\ &\sqcap \neg \exists isEqualTo.\top \\ &\sqcap \neg \exists isSmallerThan.\top \\ &\sqcap \neg \exists relationshipLinkIdentifier.\top \\ &\sqcap \neg \exists relationshipLinkNonIdentifier.\top \\ &\sqcap \neg \exists iRelationshipLinkIdentifier.\top \\ &\sqcap \neg \exists iRelationshipLinkNonIdentifier.\top \\ &\sqcap \neg \exists linkedRelationships.\top \\ &\sqcap \neg \exists differentParticipation.\top \\ &\sqcap \neg \exists smallerCardinality.\top \\ &\sqcap \forall hasPart.Node \end{aligned}$
Node	$\sqsubseteq \exists isPartOf.Schema$
Element	$\begin{aligned} &\sqsubseteq Node \\ &\sqcap \forall hasAttribute.Attribute \end{aligned}$

Attribute	$\sqsubseteq \text{Element}$ $\Box \neg \exists \text{isSpecializationOf}.\top$ $\Box \neg \exists \text{isGeneralizationOf}.\top$ $\Box \neg \exists \text{relation}.\top$ $\Box \neg \exists i\text{Relation}.\top$ $\Box \neg \exists \text{hasPart}.\top$ $\Box \neg \exists \text{isEqualTo}.\top$ $\Box \neg \exists \text{isSmallerThan}.\top$ $\Box \neg \exists \text{relationshipLinkIdentifier}.\top$ $\Box \neg \exists \text{relationshipLinkNonIdentifier}.\top$ $\Box \neg \exists i\text{RelationshipLinkIdentifier}.\top$ $\Box \neg \exists i\text{RelationshipLinkNonIdentifier}.\top$ $\Box \neg \exists \text{differentParticipation}.\top$ $\Box \neg \exists \text{smallerCardinality}.\top$ $\Box \neg \exists \text{linkedRelationships}.\top$ $\Box \forall \text{isAttributeOf}.\text{Element}$ $\Box \forall \text{isPartOf}.\text{Schema}$
DiscriminatorAttribute	$\sqsubseteq \text{Attribute}$ $\Box \neg \text{IdentifierAttribute}$
IdentifierAttribute	$\sqsubseteq \text{Attribute}$ $\Box \neg \text{DiscriminatorAttribute}$
EntityType	$\sqsubseteq \text{Element}$ $\Box \neg (\text{Attribute} \sqcup \text{Relationship})$ $\Box \neg \exists i\text{Relation}.\top$ $\Box \neg \exists \text{isAttributeOf}.\top$ $\Box \neg \exists \text{differentParticipation}.\top$ $\Box \neg \exists \text{smallerCardinality}.\top$ $\Box \neg \exists \text{linkedRelationships}.\top$ $\Box \forall \text{relation}.\text{Relationship}$ $\Box \forall \text{isPartOf}.\text{Schema}$

AssociativeEntity	$\sqsubseteq \textit{EntityType}$ $\Box \neg \exists \textit{isSpecializationOf}.\top$ $\Box \neg \exists \textit{isGeneralizationOf}.\top$ $\Box \neg \exists \textit{relationshipLinkTotal.Relationship}$ $\Box \neg \exists \textit{hasAttribute}.\top$ $\Box \neg \exists \textit{relationshipLinkIdentifier}.\top$ $\Box \neg \exists \textit{relationshipLinkNonIdentifier}.\top$ $\Box \neg \exists i \textit{RelationshipLinkIdentifier}.\top$ $\Box \neg \exists i \textit{RelationshipLinkNonIdentifier}.\top$ $\Box \neg \exists \textit{isEqualTo}.\top$ $\Box \neg \exists \textit{isSmallerThan}.\top$ $\Box = 1 \textit{hasPart.Relationship}$ $\Box \forall \textit{hasPart.Relationship}$
-------------------	--

Entity	$\sqsubseteq \text{EntityType}$ $\Box \neg \exists \text{isSpecializationOf.Self}$ $\Box \neg \exists \text{hasPart}.\top$ $\Box \neg \exists \text{isSmallerThan.Self}$ $\Box \exists \text{isEqualTo.Self}$ $\Box \neg \exists \text{dependsOn.Self}$ $\Box \neg ((\exists \text{RelationshipLinkIdentifier}.\top)$ $\Box \neg ((\exists \text{RelationshipLinkNonIdentifier}.\top)$ $\Box \neg ((\exists \text{isSpecializationOf.Entity})$ $\Box (\exists \text{relationshipLinkIdentifier.IdentifierRelationship}))$ $\Box \neg ((\exists \text{hasAttribute.DiscriminatorAttribute})$ $\Box (\exists \text{relationshipLinkIdentifier.IdentifierRelationship})$ $\Box (\exists \text{relationshipLinkTotalOne.IdentifierRelationship}))$ $\Box \neg ((\exists \text{relationshipLinkTotalOne.IdentifierRelationship})$ $\Box (\geq 2 \text{relationshipLinkIdentifier.IdentifierRelationship}))$ $\Box \neg ((\exists \text{hasAttribute.IdentifierAttribute})$ $\Box (\exists \text{isSpecializationOf.Entity}))$ $\Box \neg ((\exists \text{hasAttribute.DiscriminatorAttribute})$ $\Box (\exists \text{isSpecializationOf.Entity}))$ $\Box \forall \text{isSpecializationOf.Entity}$ $\Box \forall \text{isGeneralizationOf.Entity}$ $\Box \forall \text{isSmallerThan.Entity}$ $\Box \forall \text{isEqualTo.Entity}$
--------	---

Relationship	$\sqsubseteq \text{Element}$ $\Box \neg \exists \text{isSpecializationOf}.\top$ $\Box \neg \exists \text{isGeneralizationOf}.\top$ $\Box \neg \exists \text{relation}.\top$ $\Box \neg \exists \text{relationshipLinkIdentifier}.\top$ $\Box \neg \exists \text{relationshipLinkNonIdentifier}.\top$ $\Box \neg \exists \text{isAttributeOf}.\top$ $\Box \neg \exists \text{hasPart}.\top$ $\Box \neg \exists \text{hasAttribute}.\text{IdentifierAttribute}$ $\Box \neg \exists \text{isEqualTo}.\top$ $\Box \neg \exists \text{isSmallerThan}.\top$ $\Box \forall i \text{Relation}.\text{EntityType}$ $\Box \forall \text{isPartOf}.\text{(AssociativeEntity} \sqcup \text{Schema)}$ $\Box \forall \text{linkedRelationships}.\text{Relationship}$ $\Box \forall \text{smallerCardinality}.\text{Relationship}$ $\Box \forall \text{differentParticipation}.\text{Relationship}$
RegularRelationship	$\sqsubseteq \text{Relationship}$ $\Box \neg \text{IdentifierRelationship}$ $\Box \neg \exists .i \text{RelationshipLinkIdentifier}.\top$ $\Box \neg \exists .i \text{RelationshipLinkNonIdentifier}.\top$
IdentifierRelationship	$\sqsubseteq \text{Relationship}$ $\Box \forall i \text{RelationshipLinkIdentifier}.\text{Entity}$ $\Box \forall i \text{RelationshipLinkNonIdentifier}.\text{Entity}$ $\Box \neg (\text{RegularRelationship} \sqcup \text{UnaryRelationship} \sqcup \text{TernaryRelationship})$
RelationshipDegree	
UnaryRelationship	$\sqsubseteq \text{RelationshipDegree}$ $\Box \neg (((i \text{RelationshipLinkPartialOne}.\text{Entity}) \sqcup (\exists i \text{RelationshipLinkTotalOne}.\text{Entity})) \sqcap (\exists \text{hasAttribute}.\text{DiscriminatorAttribute}))$

BinaryRelationship	$\sqsubseteq \text{RelationshipDegree}$ $\Box \neg ((i\text{RelationshipLinkPartialOne.Entity})$ $\sqcup (\exists i\text{RelationshipLinkTotalOne.Entity}))$ $\Box (\exists \text{hasAttribute.DiscriminatorAttribute}))$
TernaryRelationship	$\sqsubseteq \text{RelationshipDegree}$
EntityNotation	
RegularEntity	$\sqsubseteq \text{EntityNotaTion}$ $\Box \neg \text{WeakEntity}$ $\Box \neg \exists \text{hasAttribute.DiscriminatorAttribute}$
WeakEntity	$\sqsubseteq \text{EntityNotaTion}$ $\Box \neg \text{RegularEntity}$ $\Box \neg \exists \text{hasAttribute.IdentifierAttribute}$

From: The author

4.4 PROOF OF CONCEPT

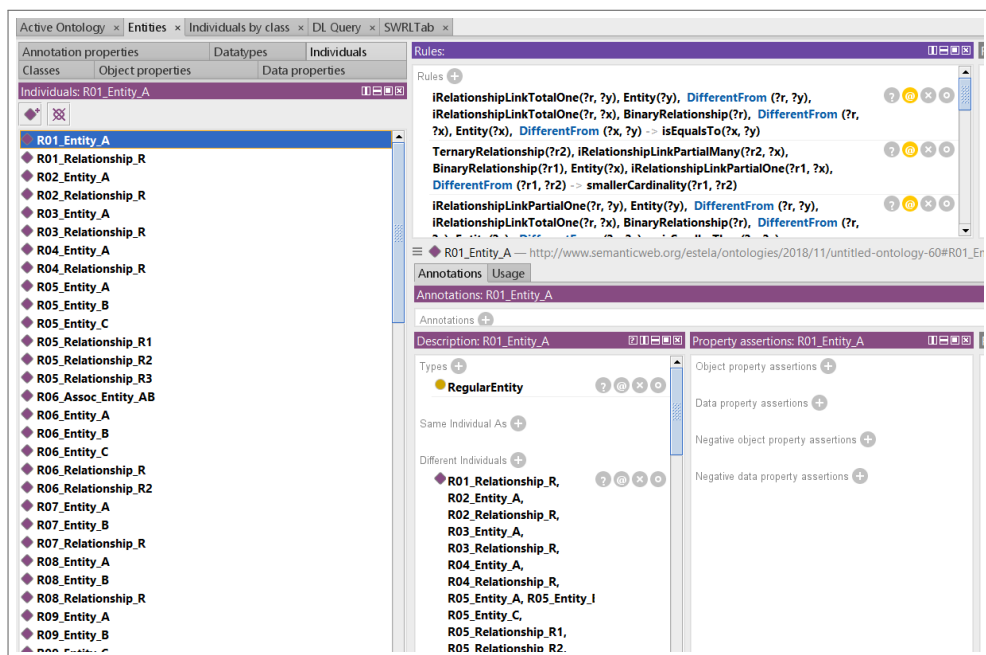
To check our Knowledge Base, we considered the constructs present in annexes A and B. Annex A consists of 27 invalid element combinations taken from Nascimento (2015). Each combination has one of the validation errors discussed in chapter 3. Annex B, on the other hand, consists of a set of valid combinations resulting from corrections made by us in each example in annex A.

Based on the sets presented, we performed the following steps:

1. We created a set of individuals in Protégé, consisting of 27 valid element combinations, according to annex B. Thus, we considered those combinations as being part of the same database conceptual model. Figure 61 shows the Individuals tab in Protégé;
2. We then used "Pellet" (SIRIN et al., 2007) (Pellet Reasoner Plug-in for Protégé, version 2.2.0) and "Hermit" (GLIMM et al., 2014) (Hermit for Protégé, version 1.3.8.413) reasoners to verify the results of the reasoning. Figures 62 and 63 present Hermit and Pellet results, respectively, highlighting the inferences for the valid combination corresponding to rule R01 (all 1:1 unary relationship types with full-partial or partial-total participation are structurally invalid) (see Figure 64). Table 16 shows the R01 example description in *ALCROIQ*;

3. Soon after, we introduced a validation error for the construct regarding the rule R01, as per annex A. Figure 65 shows the invalid construct combination used to check the R01 rule, with each element annotated with its corresponding KB concept. Table 17 shows the corresponding description in *ALCROIQ*;
4. We reran the reasoners. Figures 66 and 67 show the results from "Pellet" and "Hermit" related to the EER model in Figure 65, respectively. Figure 68 shows the "Inconsistent ontology explanation" window presented by Protégé;
5. We then restored the previous state, and repeated steps 3 and 4, introducing the errors for the rules R02 to R27, one at a time;
6. Finally, we added all the 27 validation errors and reran the reasoners to verify the inferences in a conceptual model with more than one error. Figures 69 and 70 show the results for Pellet and Hermit, respectively.

Figure 61 – Protégé individuals tab



From: The author

It is essential to highlight that we created the individuals manually. In future work, we plan to create a tool to automate the process of creating individuals from EER models.

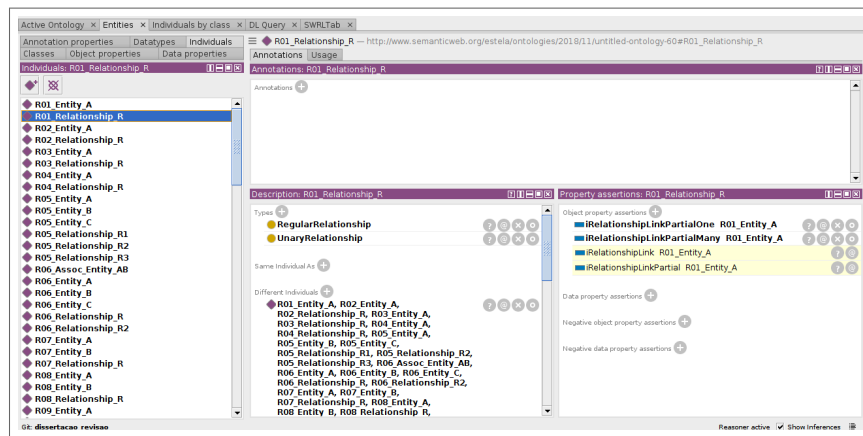
The experiments showed that the reasoners could effectively detect errors in the semantically invalid models, including models with more than one invalid structure. They also showed that the reasoners could successfully reason on the valid samples.

Table 16 – R01 valid model (ABox fragment in $\mathcal{ALCROIQ}$)

$RegularEntity(R01_Entity_A)$
$RegularRelationship(R01_Relationship_R)$
$UnaryRelationship(R01_Relationship_R)$
$iRelationshipLinkPartialOne(R01_Relationship_R, R01_Entity_A)$
$iRelationshipLinkPartialMany(R01_Relationship_R, R01_Entity_A)$

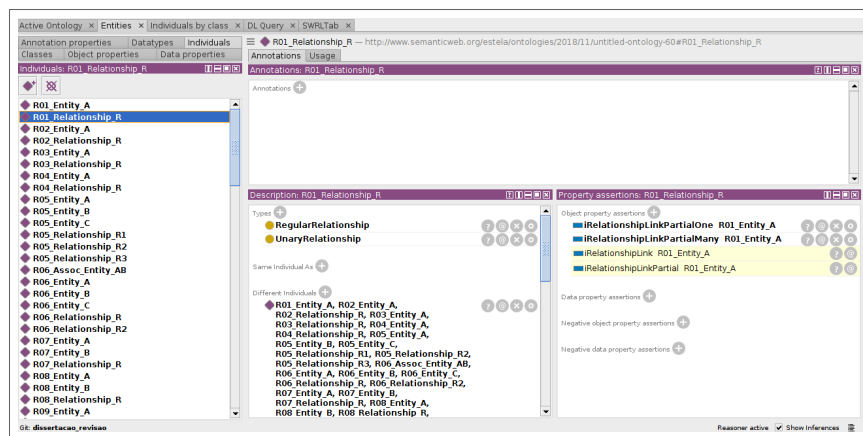
From: The author

Figure 62 – Pellet reasoning result (Protégé)



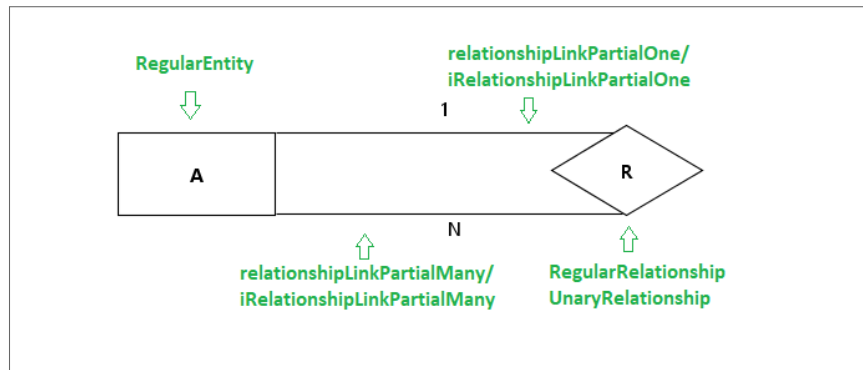
From: The author

Figure 63 – Hermit reasoning result (Protégé)



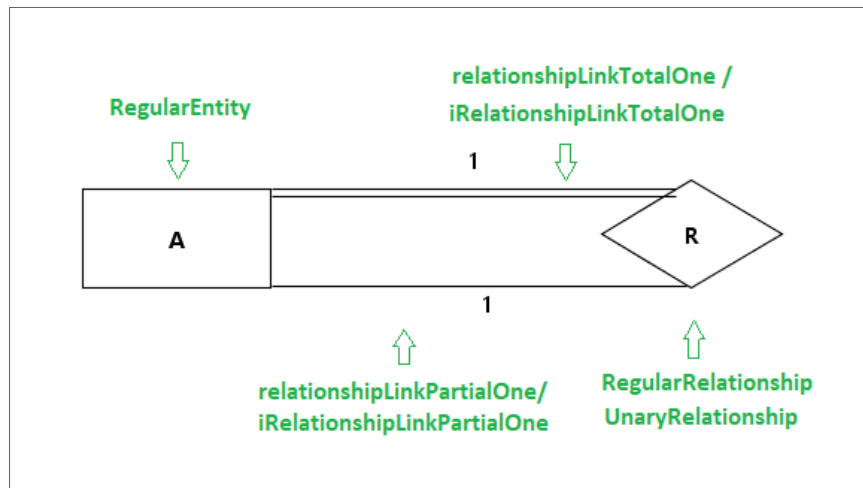
From: The author

Figure 64 – Valid model according to R01



From: The author

Figure 65 – Invalid model according to R01



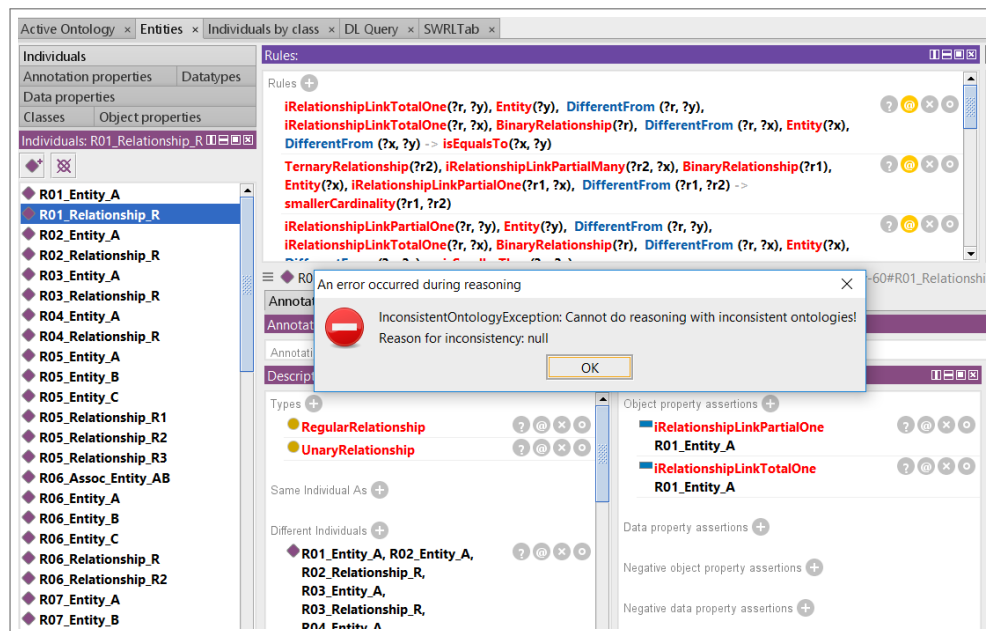
From: The author

Table 17 – R01 invalid model (ABox fragment in $\mathcal{ALCROIQ}$)

$RegularEntity(R01_Entity_A)$
$RegularRelationship(R01_Relationship_R)$
$UnaryRelationship(R01_Relationship_R)$
$iRelationshipLinkPartialOne(R01_Relationship_R, R01_Entity_A)$
$iRelationshipLinkTotalOne(R01_Relationship_R, R01_Entity_A)$

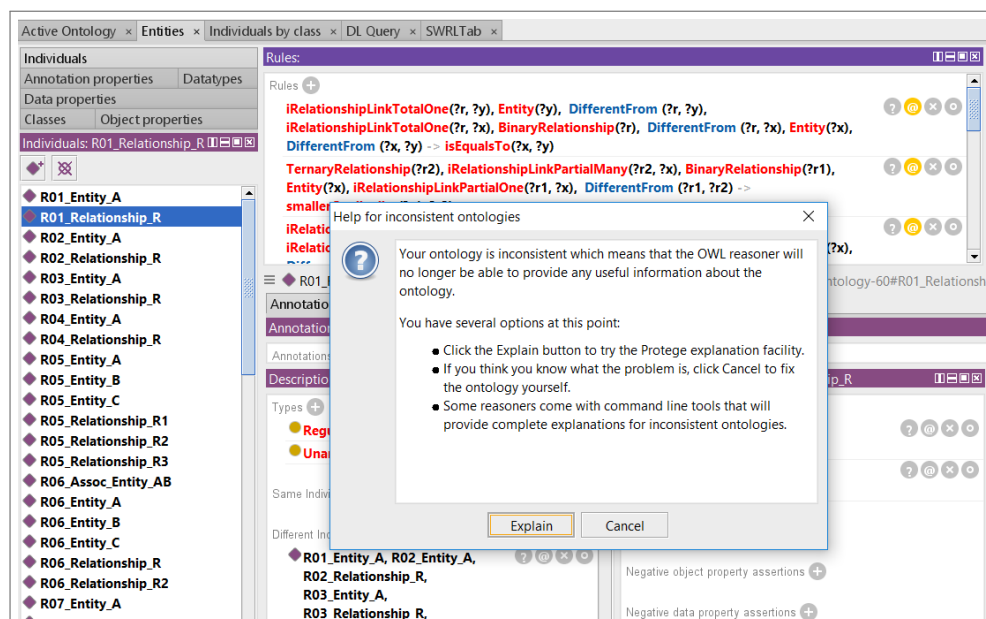
From: The author

Figure 66 – Pellet reasoning result (Protégé)



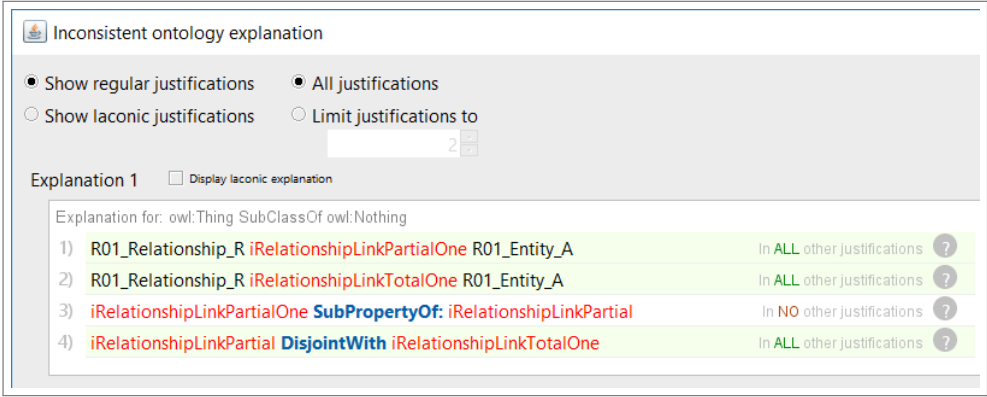
From: The author

Figure 67 – Hermit reasoning result (Protégé)



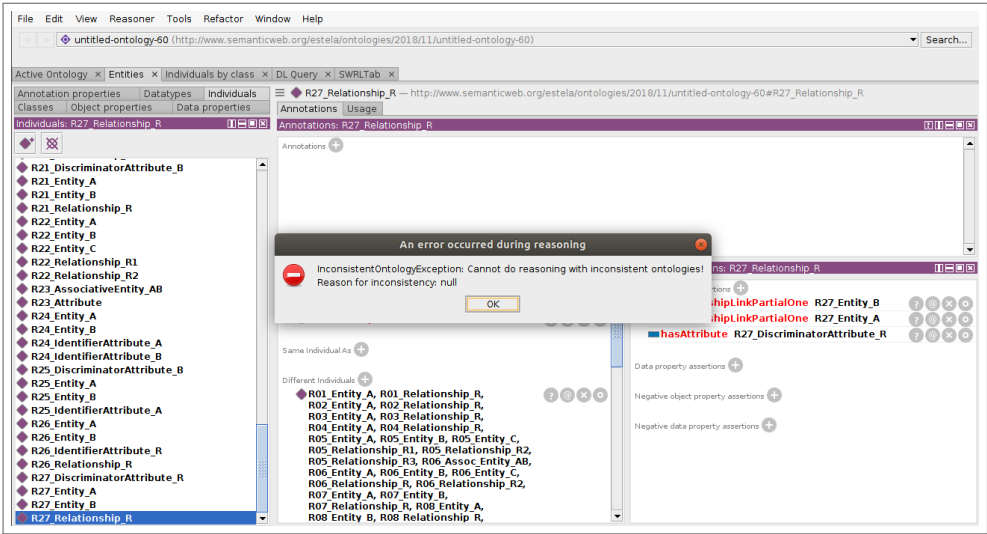
From: The author

Figure 68 – Inconsistent ontology explanation



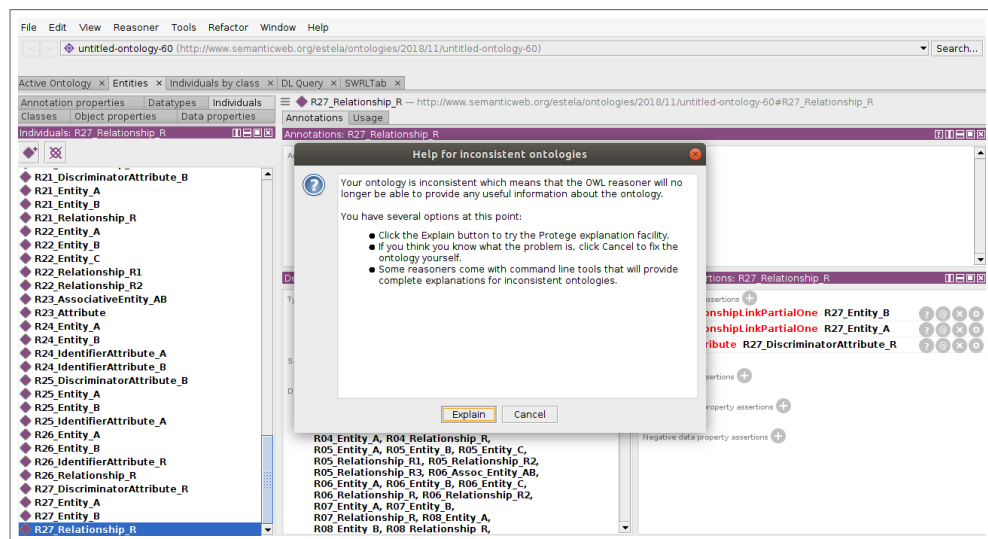
From: The author

Figure 69 – Pellet reasoning result (27 validation errors)



From: The author

Figure 70 – Hermit reasoning result (27 validation errors)



From: The author

5 RELATED WORK

5.1 INTRODUCTION

This chapter presents primary studies related to the validation of static conceptual models, briefly describing their methodologies, results, and limitations. First, we briefly present the systematic literature review conducted by Gonzalez and Cabot (2014) and the resultant primary studies related to static models validation using description logics. Next, we present the study selected after a brief survey conducted by us to identify more recent work related to the validation of Enhanced Entity-Relationship models. Finally, we highlight the main contributions of our work.

5.2 STATIC MODELS VALIDATION IN THE LITERATURE

In their literature review, Gonzalez and Cabot (2014) analyzed studies related to the formal validation of static models, i.e., models that represent structures that do not provide time dependency. UML Class Diagrams and Enhanced Entity-Relationship (EER) models are examples of static models. The authors selected 48 studies published between 2002 and 2012 and organized them into 18 groups according to their characteristics. It is necessary to mention that, during the paper selection phase, the authors found a problem regarding the absence of standard terminology when dealing with model verification. While terms such as "consistency checking" and "satisfiability" referred to disjoint concepts in some work, in other studies they related to the same property. In this work, we consider these terms as synonyms.

After analyzing the results, Gonzalez and Cabot (2014) identified that the studies focused on verifying two main types of correctness properties: properties related to model satisfiability, i.e., checking whether it is possible to create an instance of a model; and properties related to constraints, such as constraint subsumption.

They also found that there are usually two stages in model verification: formalization and reasoning. The formalization stage usually occurs through the use of a logical representation (first-order logic - FOL, description logic - DL or higher order logic - HOL) or by using a specification language such as B or Object-Z. Some studies also encode the problem as a Constraint Satisfaction Problem (CSP). The reasoning stage, on the other hand, is conducted by solvers or by specialized tools, according to the formalism adopted in the first stage.

From the formalisms presented, we can highlight Description Logics (DLs). As previously mentioned, DLs are a set of languages used for knowledge representation, allowing to represent concepts, roles, and individuals by creating Knowledge Bases (KBs). It also admits decidable and automated reasoning tasks, such as verifying whether a KB is sat-

isfiable, establishing hierarchies between concepts (subsumption) and checking concept equivalence. It also allows the identification of additional implicit logical consequences that may be hard to identify manually.

From the 48 studies selected by Gonzalez and Cabot (2014), 13 publications related to the use of DLs for the formalization of conceptual models, of which 11 focused on UML Class Diagrams and two on EER models.

Calì et al. (2002) proposed a mapping from UML Class Diagrams to TBoxes in \mathcal{DLR} . Berardi, Calvanese and Giacomo (2005) compiled the results from Berardi (2002), that proposed a mapping from UML Class Diagrams to TBoxes in \mathcal{DLR}_{ifd} , and Berardi, Calvanese and Giacomo (2003), that investigated the computational complexity related to reasoning over UML Class Diagrams. It also proposed an encoding from UML Class Diagrams to TBoxes in \mathcal{ALCQI} , in order to make use of reasoning systems.

Cadoli et al. (2004), Cadoli, Calvanese and Giacomo (2004) and Cadoli et al. (2007) discussed a problem regarding the reasoning over database and software systems conceptual models mapped to expressive DLs. They introduced the concepts of unrestricted satisfiability and finite satisfiability. According to the authors, unrestricted satisfiability concerns whether the KB admits non-empty models (finite or infinite), while finite satisfiability relates to whether the KB admits non-empty and finite models. According to Cadoli, Calvanese and Giacomo (2004), for expressive DLs, it is only possible to ensure unrestricted satisfiability. They claim that there is a lack of a finite model property in expressive DLs, which occurs "due to the interaction between cardinality constraints, the use of direct and inverse roles, and general (possibly cyclic) inclusion assertions in the knowledge base."

When dealing with databases and software systems, it is crucial to ensure a finite model since they only admit a finite number of instances (e.g., we can not have infinite tuples in a database). Consequently, when considering a mapping from databases and software systems conceptual models to an expressive DL, it is essential to use additional validation to ensure finite satisfiability. In order to achieve this goal, the authors used a technique for finite model reasoning in DLs by encoding the problem as a Constraint Satisfaction Problem (CSP).

Artale et al. (2007a) and Artale et al. (2007b) investigated the computational complexity of reasoning over various fragments of the EER language. Based on Berardi, Calvanese and Giacomo (2005), they mapped EER models to TBoxes in different DL languages. The language varied according to the chosen fragment of the EER model.

Artale, Calvanese and Ibanez-Garcia (2010a) and Artale, Calvanese and Ibanez-Garcia (2010b) investigated the computational complexity of reasoning to check full satisfiability, i.e., whether there are model instances where all classes and associations are non-empty, over various fragments of the UML Class Diagram. They mapped the diagrams to TBoxes in different DL languages, varying the expressiveness according to the chosen fragment.

Queralt et al. (2012a) and Queralt et al. (2012b) checked finite satisfiability of UML Class Diagrams enriched with OCL constraints. Since checking OCL constraints is known to be undecidable, they identified a decidable fragment of OCL, which they called OCL-Lite. They also investigated the computational complexity of the reasoning task. To ensure finite satisfiability, they encoded the models and constraints as TBoxes in *ALC_LI*, a less expressive DL that enjoys the finite model property.

Since our work focus on the formal verification of EER models, we conducted a brief survey in order to select more recent studies related to the use of DL to formalize those models. Our methodology and result will be presented in the next section, followed by our final remarks.

5.3 SURVEY

In our survey for more recent work on the formal verification of EER models, we considered only work published since 2013 and written in English, complementing the analysis made by Gonzalez and Cabot (2014). By using the search string ("description logic" and "Entity-Relationship"), we collected papers from the following repositories: ACM Digital Library, Science Direct, Springer Link, and IEEE XPlore Digital Library. Although the repositories returned 235 results, we only could access 120 of them by using the university's network.

We established two phases for the analysis of the work. In the first phase, we analyzed their title and abstract and separated them into three groups: "approved", "excluded" and "for further analysis". In the second phase, we examined the introduction and the conclusion of the studies from the "for further analysis" group. Table 18 shows a summary of the results of each phase. In the end, we only selected the research work by Zhang, Ma and Cheng (2016). A small number of studies was already expected, based on the outcomes from Gonzalez and Cabot (2014).

Table 18 – Survey results summary

Repositories	Retrieved	Accessed	Phase 1	Phase 2
ACM Digital Library	1	1	0	0
Science Direct	57	51	5	1
Springer Link	150	43	3	0
IEEE XPlore Digital Library	27	25	2	0
Total	235	120	10	1

From: The author

Zhang, Ma and Cheng (2016) proposed a mapping from EER models to TBoxes in *ALC_LQIK*. They also presented a prototype tool, the EER2DL, responsible for automating the mapping process by taking models in *.xmi format from some CASE tools and creating a KB. From the output of EER2DL, they could use automatic reasoners to perform syntax

checking. It is important to note that, unlike the UML language family, the EER language does not have a standard metamodel, so the structure of the *.xmi files varies according to the metamodel used by each modeling tool. Based on that, although it was not explicitly exposed, the approach taken by Zhang, Ma and Cheng (2016) depends on the metamodels used by each tool to perform the transformation of models into KBs. As stated before, our study considers the EERMM metamodel.

5.4 FINAL REMARKS

All related work presented in this chapter used Description Logics to formalize static conceptual models. They mapped models to TBoxes in multiple DL languages aiming to explore the results provided by automatic reasoners. The automatic verification provided by those tools included the following tasks: schema satisfiability, entity/class consistency (or satisfiability), entity/class equivalence, entity/class subsumption, and the identification of implicit logical consequences. Some work focused on ensuring finite satisfiability (which is not guaranteed by expressive DLs) either by using less expressive DLs or by adopting additional tools.

Also, few work have focused on the Enhanced Entity-Relationship model. While Artale et al. (2007a) and Artale et al. (2007b) have centered on identifying the computational complexity of reasoning on EER models represented as a TBox, Zhang, Ma and Cheng (2016) focused on limited syntactic and semantic aspects, not covering validations related to the interaction between model constructors and constraints.

Therefore, to the best of our knowledge, our work stands out for considering the formalization of the EER language to model validation, covering aspects not addressed in other work, such as the consequences of constraints such as cardinality, participation, relationship type degree, inheritance, cyclic paths, and valid attribute types, as well as the interactions between these concepts in the same model.

6 CONCLUSION

6.1 INTRODUCTION

In this chapter, we summarize the project steps, recalling the methodology used, and the results obtained. We also present recommendations for future work.

6.2 FINAL REMARKS

The proper validation of conceptual models is crucial in a database project since unidentified conceptual errors can be disseminated to the other phases of the project, negatively affecting the results. Although on small projects, designers can quickly perform the validation manually, on large projects, the support of automated tools becomes necessary.

To create conceptual models, designers use modeling languages. The Enhanced Entity-Relationship (EER) language is commonly used in academia for conceptual database modeling, adding the concepts of superclass, subclass, and category to the Entity-Relationship language (Chen, 1976). Since EER is a language, the validity of the models created from it is directly influenced by the proper application of its syntactic and semantic rules.

Description Logics (DLs) are a set of languages used for knowledge representation, which admit decidable and automated reasoning tasks such as the identification of implicit logical consequences that may be hard to identify manually. Because of those features, DLs have been recognized as a promising alternative to represent and reason on conceptual models ((ZHANG; MA; CHENG, 2016), (ARTALE; CALVANESE; IBANEZ-GARCIA, 2010a), (BERARDI; CALVANESE; GIACOMO, 2005)).

Based on that, the main goal of this work was to support the automatic validation of conceptual database models by identifying syntactic and semantic errors in EER models using DL reasoners.

To achieve our goals, we performed the following activities:

- **Verification of the coverage of related work:** we found that few studies focus on the validation of EER models using DL. Also, they do not cover semantic aspects such as the consequences of the interaction between cardinality, participation, relationship type degree, inheritance, cyclic paths, and valid attribute types;
- **Formalization of EER syntax and semantics:** since our model validation is based on the language rules, we first formalized the EER syntax and semantics in OWL DL, creating a TBox. As the EER language does not have a standard abstract syntax, we considered the EERMM, which covers all EER constructs (Fidalgo et al., 2013). The semantic rules formalized are part of a catalog organized by Nascimento (2015) and based on the work of Dullea, Song and Lamprou (2003) and Calvanese

and Lenzerini (1994). Although we tried to represent most of the rules by using axioms, we also made use of Semantic Web Rule Language (SWRL) rules in the cases involving cyclic paths, since DL expressivity was not adequate;

- **Proof of concept:** After that, we created individuals (ABox) based on a test case set composed of valid and invalid models and used DL reasoners to identify the invalid constructs;
- **DL description:** finally, we manually converted the KB to *ALCROIQ*.

The experiments showed that the reasoners could effectively detect errors related to cardinality, participation, relationship type degree, inheritance, cyclic paths, valid attribute types, and the interactions between these constraints. Hence, the main contributions of our work are:

- The automatic support for validating EER models, covering aspects not handled by other projects;
- The formalization of the EER language through DL, which allows checking the ER model structure with the use of DL reasoners;
- The possibility of reusing the Knowledge Base if new validity rules need to be added or in case of a language extension;
- The formalization of the EERMM, which contributes to a possible standardization of the EER abstract syntax.

As mentioned, this work considers the EER language semantic rules related to cardinality and participation constraints, relationship types degrees, cyclic paths, and attribute types. Although the rules provide good coverage concerning the elements and constraints of the EER language, they do not cover all possible combinations. The methodology used to obtain the rules analyzed a set of constructors and their interactions in isolation and did not thoroughly evaluate the consequences of the interactions between multiple sets of constructors in the same model. Additionally, the rules do not cover relationship types degrees above 4, nor the category constructor, because of their restricted use.

6.3 FUTURE WORK

For future work, we propose the following projects:

- To develop a tool to transform EER models into an ABox, automatically;
- To evaluate models from industry applications;
- To integrate our KB into a modeling tool to help designers detect validity errors;

- To add semantic validity rules related to the category concept, which was omitted because of its restricted use.

REFERENCES

- ARTALE, A.; CALVANESE, D.; IBANEZ-GARCIA, A. Checking full satisfiability of conceptual models. p. 12, 2010.
- ARTALE, A.; CALVANESE, D.; IBANEZ-GARCIA, A. Full Satisfiability of UML Class Diagrams. In: PARSONS, J.; SAEKI, M.; SHOVAL, P.; WOO, C.; WAND, Y. (Ed.). *Conceptual Modeling – ER 2010*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. v. 6412, p. 317–331. ISBN 978-3-642-16372-2 978-3-642-16373-9. Available at: <http://link.springer.com/10.1007/978-3-642-16373-9_23>.
- ARTALE, A.; CALVANESE, D.; KONTCHAKOV, R.; RYZHIKOV, V.; ZAKHARYASCHEV, M. Complexity of Reasoning in Entity Relationship Models. p. 16, 2007.
- ARTALE, A.; CALVANESE, D.; KONTCHAKOV, R.; RYZHIKOV, V.; ZAKHARYASCHEV, M. Reasoning over Extended ER Models. In: PARENT, C.; SCHEWE, K.-D.; STOREY, V. C.; THALHEIM, B. (Ed.). *Conceptual Modeling - ER 2007*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. v. 4801, p. 277–292. ISBN 978-3-540-75562-3 978-3-540-75563-0. Available at: <http://link.springer.com/10.1007/978-3-540-75563-0_20>.
- BAADER, F. *The Description Logic Handbook: Theory, Implementation and Applications*. [S.l.]: Cambridge University Press, 2003. Google-Books-ID: e6_hJtM07qwC. ISBN 978-0-521-78176-3.
- BAADER, F.; HORROCKS, I.; SATTTLER, U. Description logics. In: _____. *Handbook on Ontologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 3–28. ISBN 978-3-540-24750-0. Available at: <https://doi.org/10.1007/978-3-540-24750-0_1>.
- BECHHOFFER, S.; HARMELEN, F. v.; HENDLER, J.; HORROCKS, I.; MCGUINNESS, D. L.; PATEL-SCHNEIDER, P. F.; STEINM, L. A. OWL Web Ontology Language Reference. p. 80, 2004.
- BERARDI, D. Using DLs to reason on UML class diagrams. *In Proc. Workshop on Applications of Description Logics*, p. 1–11, 2002.
- BERARDI, D.; CALVANESE, D.; GIACOMO, G. D. Reasoning on UML Class Diagrams is EXPTIME-hard. p. 10, 2003.
- BERARDI, D.; CALVANESE, D.; GIACOMO, G. D. Reasoning on UML class diagrams. *Artificial Intelligence*, v. 168, n. 1-2, p. 70–118, Oct. 2005. ISSN 00043702. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S0004370205000792>>.
- BRAMBILLA, M.; CABOT, J.; WIMMER, M. *Model-Driven Software Engineering in Practice*. [S.l.]: Morgan & Claypool, 2012.
- CADOLI, M.; CALVANESE, D.; GIACOMO, G. D. Towards Implementing Finite Model Reasoning in Description Logics. p. 9, 2004.
- CADOLI, M.; CALVANESE, D.; GIACOMO, G. D.; MANCINI, T. Finite satisfiability of uml class diagrams by constraint programming. p. 15, 2004.

CADOLI, M.; CALVANESE, D.; GIACOMO, G. D.; MANCINI, T. Finite Model Reasoning on UML Class Diagrams Via Constraint Programming. In: BASILI, R.; PAZIENZA, M. T. (Ed.). *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. v. 4733, p. 36–47. ISBN 978-3-540-74781-9 978-3-540-74782-6. Available at: <http://link.springer.com/10.1007/978-3-540-74782-6_5>.

CALVANESE, D.; LENZERINI, M. On the Interaction Between ISA and Cardinality Constraints. In: *Proceedings of the Tenth International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1994. p. 204–213. ISBN 978-0-8186-5400-8. Available at: <<http://dl.acm.org/citation.cfm?id=645479.655108>>.

CALÌ, A.; CALVANESE, D.; GIACOMO, G. D.; LENZERINI, M. A Formal Framework for Reasoning on UML Class Diagrams. In: GOOS, G.; HARTMANIS, J.; LEEUWEN, J. van; HACID, M.-S.; RAŚ, Z. W.; ZIGHED, D. A.; KODRATOFF, Y. (Ed.). *Foundations of Intelligent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. v. 2366, p. 503–513. ISBN 978-3-540-43785-7 978-3-540-48050-1. Available at: <http://link.springer.com/10.1007/3-540-48050-1_54>.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, v. 1, n. 1, p. 9–36, 1976.

CONNOLLY, T.; BEGG, C. Database systems. a practical approach to design, implementation and management. In: _____. [S.l.: s.n.], 2010.

DULLEA, J.; SONG, I.-Y.; LAMPROU, I. An analysis of structural validity in entity-relationship modeling. *Data & Knowledge Engineering*, v. 47, n. 2, p. 167–205, Nov. 2003. ISSN 0169023X. Available at: <<http://linkinghub.elsevier.com/retrieve/pii/S0169023X03000491>>.

ELMASRI, R.; NAVATHE, S. *Fundamentals of Database Systems*. 6th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0136086209, 9780136086208.

FIDALGO, R. D. N.; SOUZA, E. M. D.; ESPAÑA, S.; CASTRO, J. B. D.; PASTOR, O. EERMM: A Metamodel for the Enhanced Entity-Relationship Model. In: *Conceptual Modeling*. Springer, Berlin, Heidelberg, 2012. (Lecture Notes in Computer Science), p. 515–524. ISBN 978-3-642-34001-7 978-3-642-34002-4. Available at: <https://link.springer.com/chapter/10.1007/978-3-642-34002-4_40>.

FIDALGO, R. N.; ALVES, E.; ESPAÑA, S.; CASTRO, J.; PASTOR, O. Metamodeling the Enhanced Entity-Relationship Model. *Journal of Information and Data Management*, v. 4, n. 3, p. 406, Sep. 2013. ISSN 21787107. Available at: <<https://seer.ufmg.br/index.php/jidm/article/view/260>>.

GLIMM, B.; HORROCKS, I.; MOTIK, B.; STOILLOS, G.; WANG, Z. Hermit: An owl 2 reasoner. *J. Autom. Reason.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 53, n. 3, p. 245–269, Oct. 2014. ISSN 0168-7433. Available at: <<http://dx.doi.org/10.1007/s10817-014-9305-1>>.

GONZALEZ, C. A.; CABOT, J. Formal verification of static software models in MDE: A systematic review. *Information and Software Technology*, v. 56, n. 8, p. 821–838, Aug. 2014. ISSN 09505849.

GRONBACK, R. *Eclipse Modeling Project / The Eclipse Foundation*. 2017. Available at: <<https://www.eclipse.org/modeling/emf/>>.

HORRIDGE, M. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3. p. 108, 2011.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOFF, B.; DEAN, M. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. 2004. Available at: <<https://www.w3.org/Submission/SWRL/>>.

KOLP, M.; ZIMANYI, E. Enhanced ER to relational mapping and interrelational normalization. *Information and Software Technology*, v. 42, n. 15, p. 1057–1073, Dec. 2000. ISSN 09505849. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S0950584900001324>>.

KROTZSCH, M.; SIMANCIK, F.; HORROCKS, I. A Description Logic Primer. *arXiv:1201.4089 [cs]*, Jan. 2012. ArXiv: 1201.4089. Available at: <<http://arxiv.org/abs/1201.4089>>.

MOE, R. E. Distilled entity relationship model. 01 2004.

MUSEN, M. A. The Protégé Project: A Look Back and a Look Forward. *AI matters*, v. 1, n. 4, p. 4–12, Jun. 2015. ISSN 2372-3483. Available at: <<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4883684/>>.

NASCIMENTO, A. J. d. *Um Catálogo de Regras para Validações Estruturais de diagramas EER*. Master's Thesis (Master's Thesis) — Universidade Federal de Pernambuco, Recife, 2015.

QUERALT, A.; ARTALE, A.; CALVANESE, D.; TENIENTE, E. OCL-Lite: A Decidable (Yet Expressive) Fragment of OCL. p. 11, 2012.

QUERALT, A.; ARTALE, A.; CALVANESE, D.; TENIENTE, E. OCL-Lite: Finite reasoning on UML/OCL conceptual schemas. *Data & Knowledge Engineering*, v. 73, p. 1–22, Mar. 2012. ISSN 0169023X. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S0169023X11001273>>.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. *Database Systems Concepts*. 5. ed. New York, NY, USA: McGraw-Hill, Inc., 2006. ISBN 0072958863, 9780072958867.

SINGH, S. K. *Database Systems: Concepts, Design & Applications*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 8177585673, 9788177585674.

SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; KATZ, Y. Pellet: A Practical OWL-DL Reasoner. p. 26, 2007.

SNOECK, M. The Existence-Dependency Graph. In: *Enterprise Information Systems Engineering*. Cham: Springer International Publishing, 2014. p. 79–105.

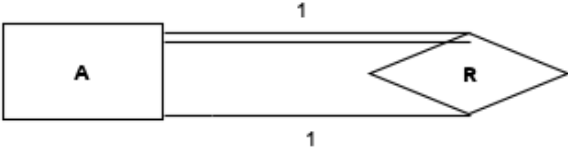
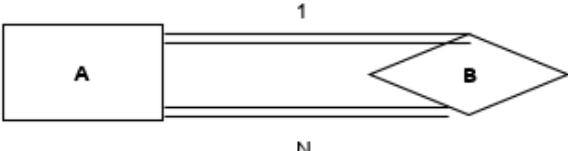
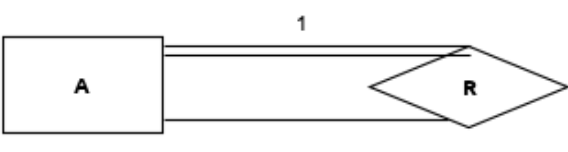
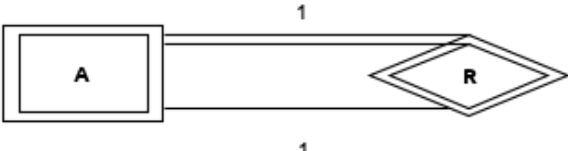
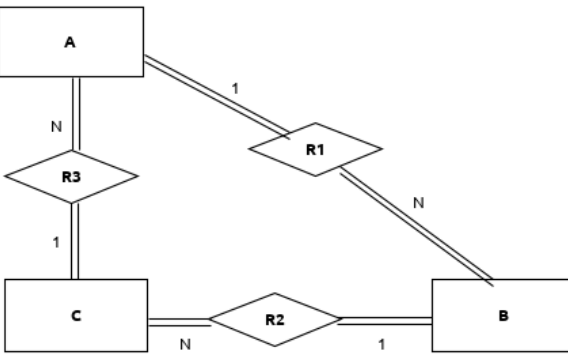
VAGNER, A. Store and Visualize EER in Neo4j. In: *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control - ISCSIC '18*. Stockholm, Sweden: ACM Press, 2018. p. 1–6. ISBN 978-1-4503-6628-1. Available at: <<http://dl.acm.org/citation.cfm?doid=3284557.3284694>>.

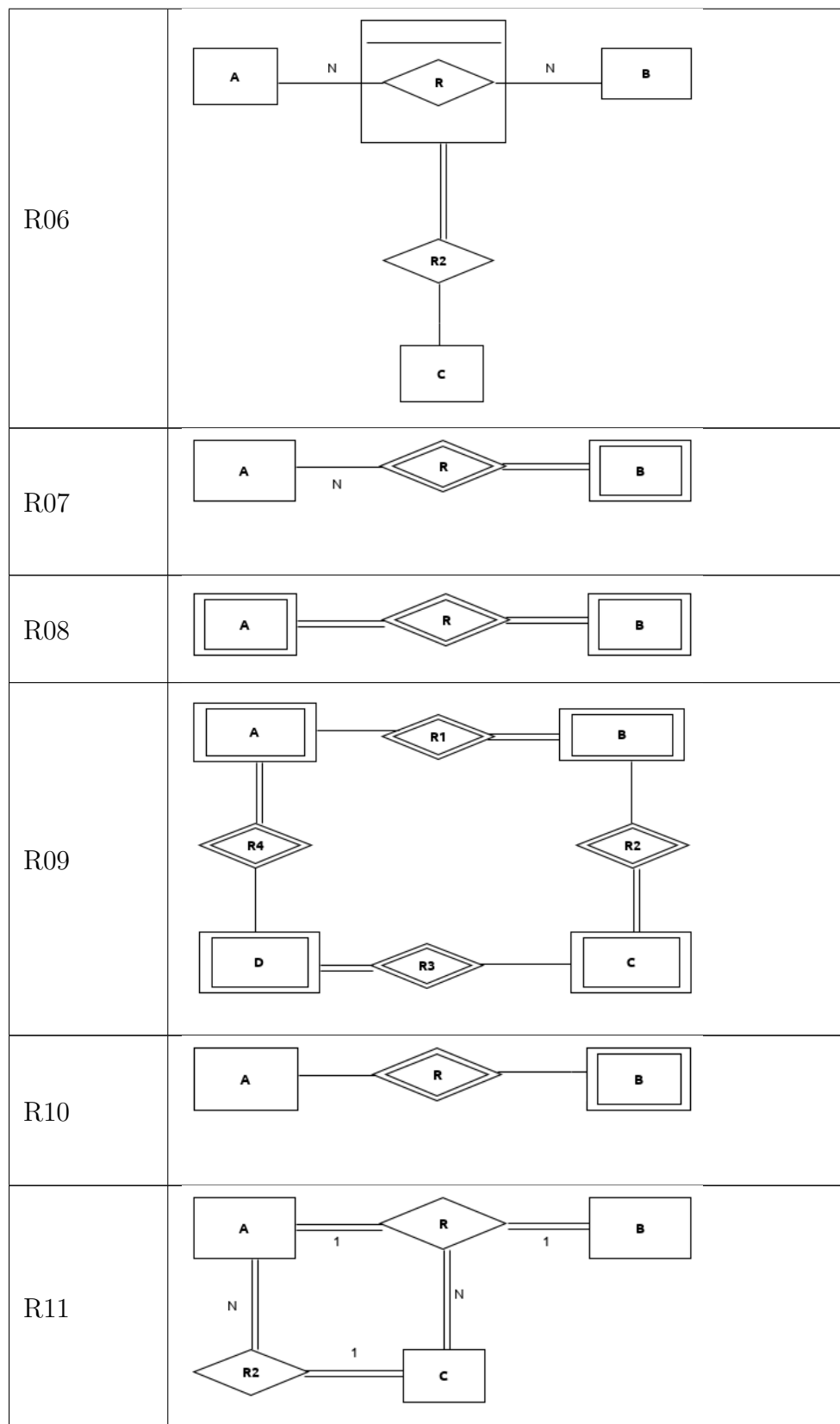
WESTLAND, J. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, v. 62, p. 1–9, May 2002. ISSN 01641212.

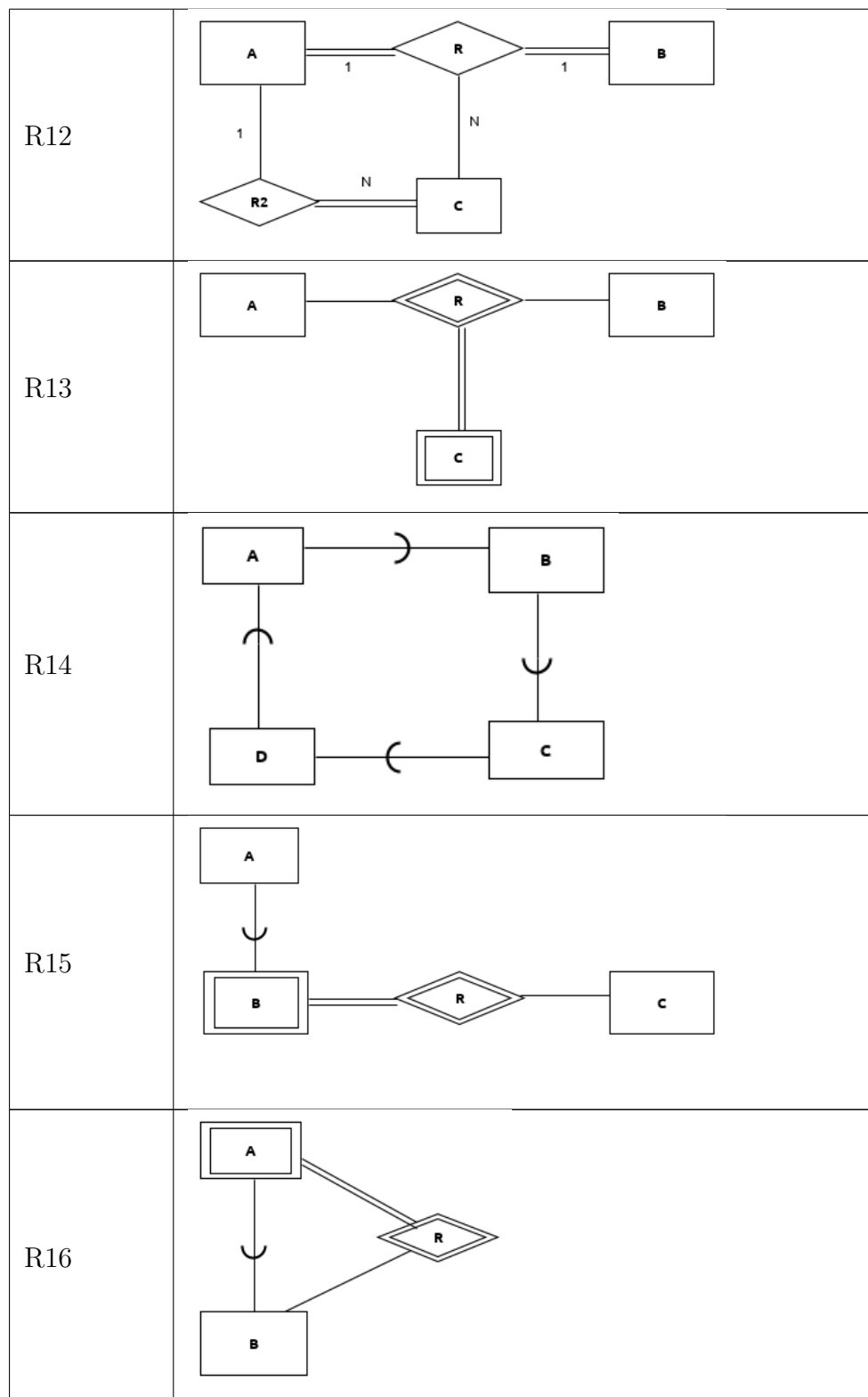
ZHANG, F.; MA, Z.; CHENG, J. Enhanced entity-relationship modeling with description logic. *Knowledge-Based Systems*, v. 93, p. 12–32, Feb. 2016. ISSN 09507051. Available at: <<https://linkinghub.elsevier.com/retrieve/pii/S0950705115004177>>.

APPENDIX A – LIST OF INVALID EXAMPLES

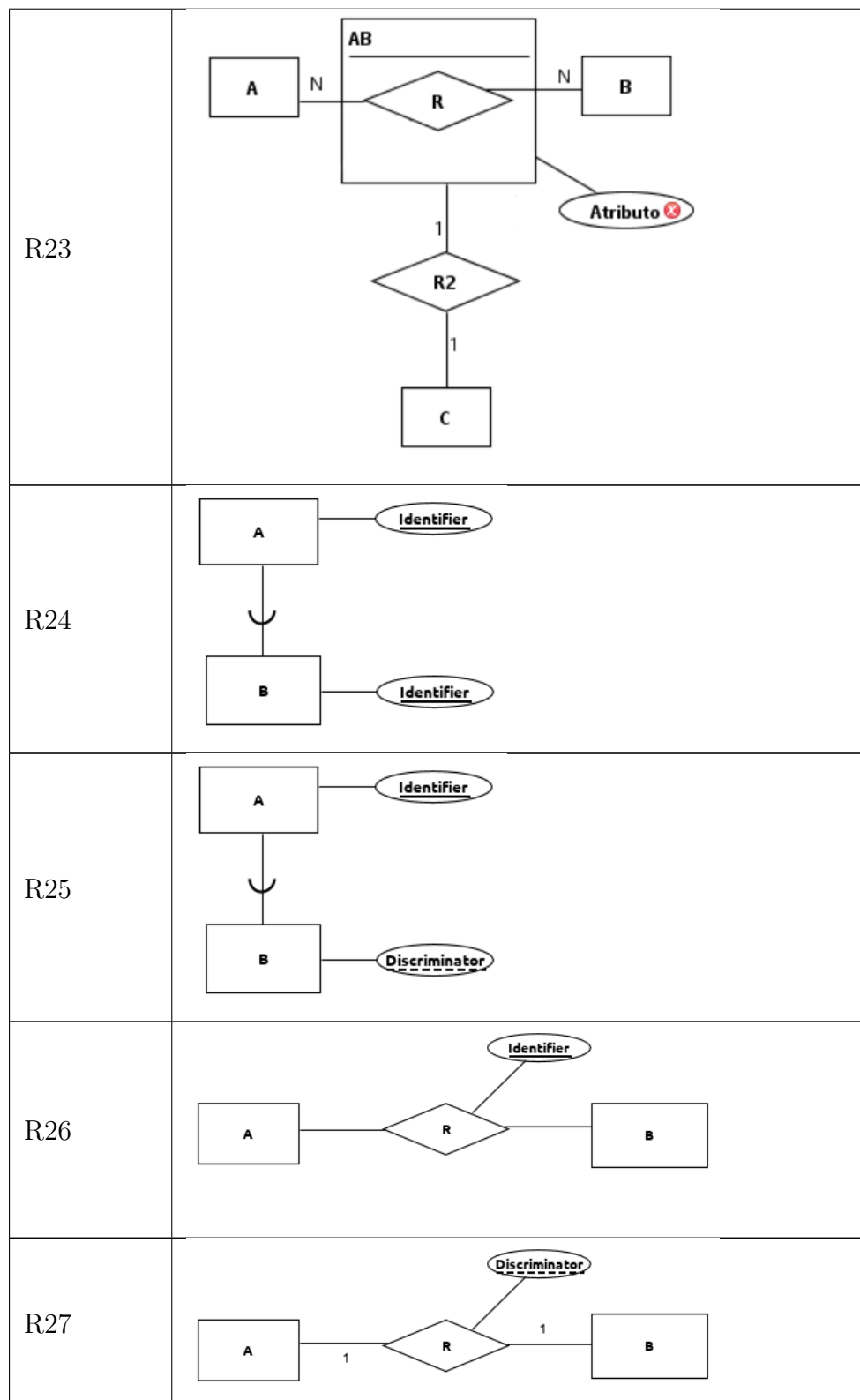
Table 19 – Invalid examples

Rule	Model
R01	
R02	
R03	
R04	
R05	





R17	
R18	
R19	
R20	
R21	
R22	




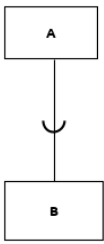
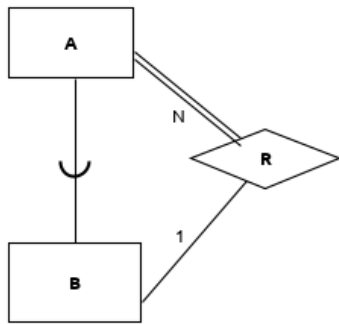
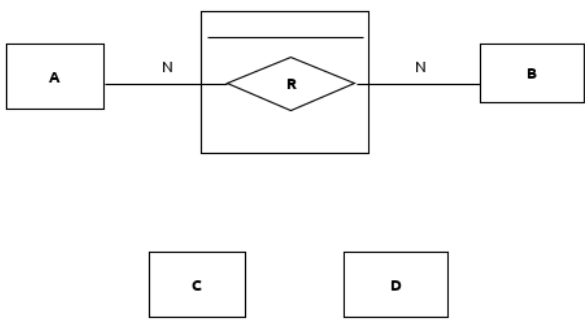
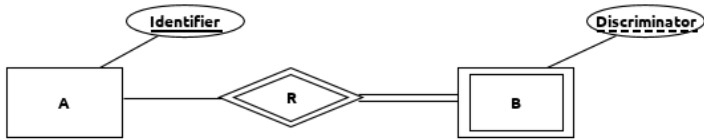
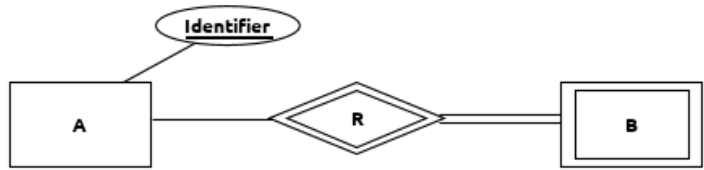
Adapted from: Nascimento (2015)

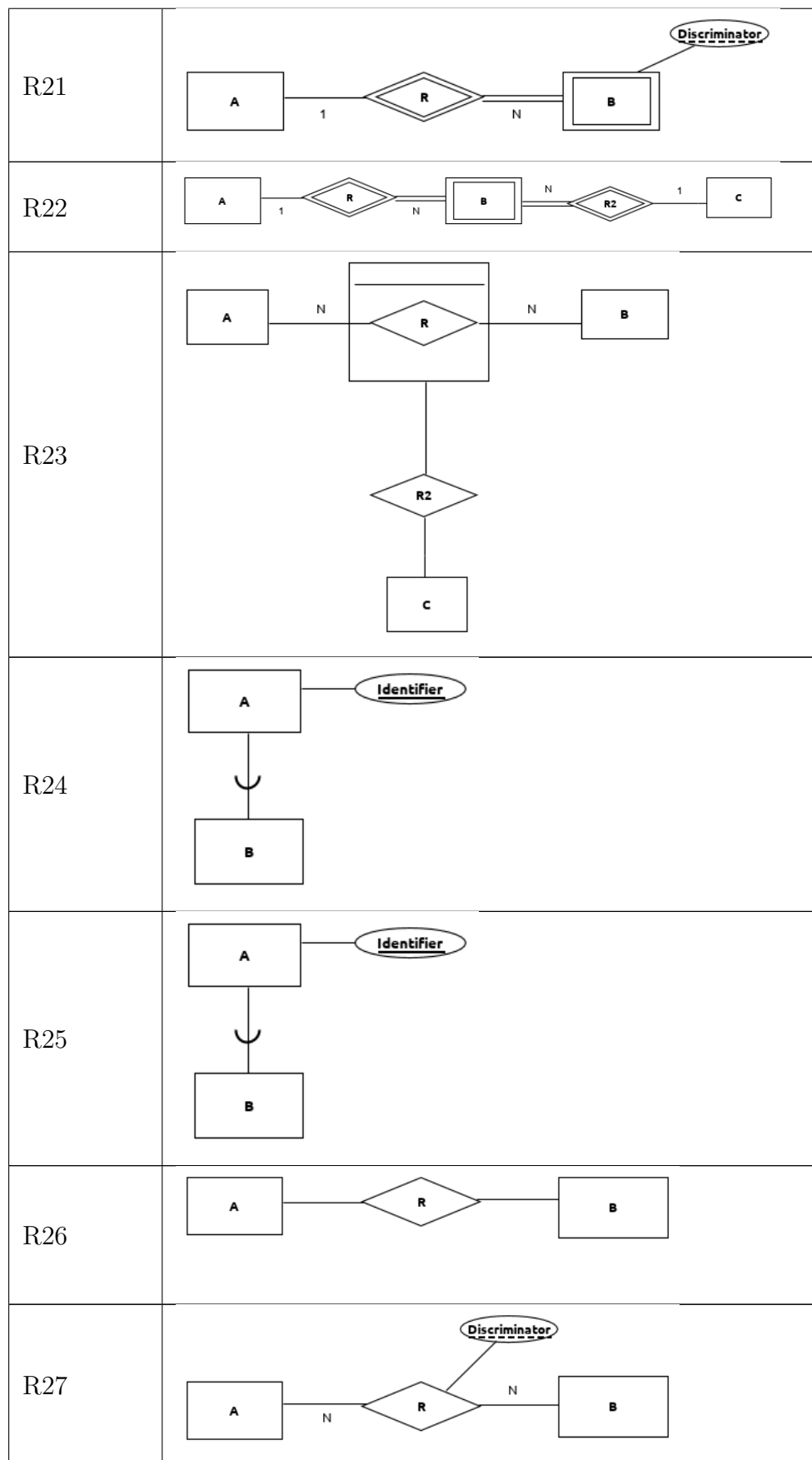
APPENDIX B – LIST OF VALID EXAMPLES

Table 20 – Valid examples

Rule	Model
R0[1-4]	
R05	
R06	
R07	
R08	

R09	<p>ER diagram R09: A double-lined rectangle A is connected to a double-lined diamond R1, which is connected to a double-lined rectangle B. A is also connected to a double-lined diamond R4, which is connected to a double-lined rectangle D. D is connected to a double-lined diamond R3, which is connected to a rectangle C.</p>
R10	<p>ER diagram R10: A rectangle A is connected to a double-lined diamond R, which is connected to a double-lined rectangle B.</p>
R11	<p>ER diagram R11: A rectangle A is connected to a diamond R with a "1" label. R is connected to a rectangle B with a "1" label. A is connected to a diamond R2 with an "N" label. R2 is connected to a rectangle C with an "N" label. R is also connected to C with an "N" label.</p>
R12	<p>ER diagram R12: A rectangle A is connected to a diamond R with a "1" label. R is connected to a rectangle B with a "1" label. A is connected to a diamond R2 with a "1" label. R2 is connected to a rectangle C with an "N" label. R is also connected to C with an "N" label.</p>
R13	<p>ER diagram R13: A rectangle A is connected to a diamond R, which is connected to a rectangle B. R is also connected to a rectangle C.</p>
R14	<p>ER diagram R14: A rectangle A is connected to a rectangle B with a crow's foot notation. A is also connected to a rectangle D with a crow's foot notation. D is connected to a rectangle C with a crow's foot notation.</p>

R15	
R16	
R17	
R18	
R19	
R20	



From: The author