TECHNISCHE UNIVERSITÄT
ILMENAU

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Fachgebiet für Verteilte Systeme und Betriebssysteme

Masters thesis

Graphical Specification Language for the Entity-Labeling Aspect

Submitted by:
Philipp Schwetschenau

Supervisor:	Prof. Dr.-Ing. habil. Winfried E. Kühnhauser
Supervisor:	Dipl.-Inf. Peter Amthor
Studies:	Computer science
Matriculation no.:	46756
Submission date:	Ilmenau, 29. November 2018

Contents

List of Figures

CHAPTER 1
Introduction

1.1 Domain

With the increasing number of IT systems, securing these systems became an obvious and important issue. For this purpose many security models and model families were developed for a wide field of application domains over the last years.

Formal security models offer possibilities to analyze them concerning security properties. However, because quantity and variety of these models grow just as much as their relevance for security-critical applications the model-based security engineering process became more complex and therefore error-prone to human deviations.

Amthor [2018] proposed a new approach called Aspect-oriented Security Engineering (AOSE) which claims to close semantic gaps between steps in the security engineering process (requirements, informal policy, formal model) to reduce the potential impact of human errors. This approach roughly adopts the idea of the aspect-oriented programming paradigm. It tailors all steps to aspects, which are non-functional requirements of the engineering process like determining requirements of policy semantics or analyzing certain security goals. There are two major classes for aspects regarding AOSE: related to policy semantics and to policy analysis.

One possible aspect of the former is the Entity Labeling Aspect (EL). It is designed to formally specify policy semantics typically found in operating systems and middleware systems. Therefore it bridges the gap and supports the transformation between informal policy and formal model. EL classifies model components into six semantic categories. The notation is on a mathematical basis and uses concepts like sets, assignments or constraints.

1.2 Motivation

The goal of reducing the impact of human errors by closing semantic gaps as much as possible with the help of AOSE/EL requires handling another formal notation, which is, in this case, EL itself and its classification into the six semantic categories. To support working with this approach, especially for the communication between different groups of people involved like model engineers, security architects, software developers or future administrators, who have to cooperate and coordinate and all have different levels of experience, Amthor [2018] considers a graphical representation to be helpful to enhance the transition between informal and formalized notation.

Amthor [2018] already uses visual representations to illustrate examples. However, the representations are not described in detail as well as they are inconsistent and ambiguous regarding several parts of the formalization (e.g. arrows can have several meanings and there is no way to specify functions with multiple input parameters). So these visualizations are appropriate to underline and support the engineering of an already formalized policy after EL-based model engineering, but are not suitable for independent modeling on a stand-alone level, because they are not equivalent to their formalized mathematical counterpart.

There are two types of visualization Amthor [2018] uses, which have different levels of abstraction: One is based on the actual model components and visualizes them and their relationships. The other one is based on a higher level of abstraction and visualizes the EL with its structure of semantic categories and their relationships.

There are other visual notations like UML or ERD, which have in common that they are tailored to particular needs in special application domains, so they can not be applied here, but may give inspiration on how to model certain semantics and relationships.

Eventually to be able to independently and visually model and work on EL-based policies there is need for an unambiguous formal graphical specification language.

1.3 Goal

The main goal is to develop a graphical specification language for EL-based security polices. This should focus on clean and unambiguous semantics of the language.

The use of the language should be as simple as possible and as comprehensive as needed. Therefore its appearance and elements should be clear and well-structured. A selection of appropriate symbols and geometrical shapes for their equivalent model counterparts has to be made regarding an intuitive understanding and workflow. This selection should not be designed contradictory or conflicting to already established notations, especially those, which may be used in the context of security engineering.

It should be evaluated how feasible and reasonable it is to develop equivalent counterparts for every possible element and relationship in context of EL and to display all of them at once. This may also lead to the question how the visualization is related to a visualization on a higher abstraction level and how they are connected and might be managed. The latter may be investigated as an optional goal.

In addition to that a GUI-based editor should be developed as a prototype to make use of the proposed graphical specification language.

1.4 Structure

CHAPTER 2 Fundamentals

This chapter provides information to all fundamentals necessary for this thesis. It will serve as basis for all design decisions in the following chapter 3 and chapter 4.

Section 2.1 introduces the RBAC security model. Section 2.2 focuses on the Aspect-oriented Security Engineering (AOSE) proposed by Amthor [2018]. It covers the Entity-Labeling Aspect in particular, one aspect of AOSE, which is going to be essential regarding the task to design a graphical specification language for it in chapter 3. In section 2.3 the well-established graphical notations UML and ER are described. Based on human optical perception and gestalt laws section 2.4 provides information on how we perceive and evaluate visual impressions regarding two-dimensional forms and structures. In the last section 2.5 information on how to design a software graphical user interface are given with respect to common best practices like design patterns and modern usability.

2.1 Security models

A security policy is a set of rules to fulfill security related requirements of an IT system. In order to be able to work with such a security policy and to analyze it, it has to be formalized as an instance of a security model. Being the central artifact in the process of model-based security engineering we want to describe a very basic security model – the Identity-based Access Control (IBAC) model – and a well-established and widely used extension of it – the Role-based Access Control (RBAC) model. In the course of this work, we need to construct security policies and models to explain and visualize the security engineering in context of the Entity-Labeling aspect. The models going to be used are RBAC models.

2.1.1 Identity-based Access Control

The most fundamental security model is probably the Identity-based Access Control (IBAC) model. The model is based on a system with active (subjects) and passive entities (objects). With IBAC requests are managed according to the requester's unique identification [Lampson, 1974]. A set of rules (*policy*) determines whether a request is permitted or forbidden. These rules can be formulated in form of an access control matrix (ACM) or an access control function (ACF). An ACF is defined as following [Amthor, 2018]:

Definition 2.1 (Access control function).
An access control function (ACF) is a function $acf : S \times O \times OP \rightarrow B$, where

- S is a set of *subject identifiers*
- O is a set of *object identifiers*
- OP is a set of *operation identifiers*

For any $s \in S, o \in O, op \in OP$, we say s is allowed to execute op on o iff $acf(s, o, op)$.

2.1.2 Role-based Access Control

With an increasing number of subjects and objects IBAC's ACFs or ACMs get very large and hard to manage effectively. Role-based Access Control models (RBAC) counter this scalability problem by using one stage of indirection. For this RBAC models extend IBAC with the concept of roles [Sandhu et al., 1996]. This design decision is based on the finding of Sandhu et al. [1996] that many subjects share the same access privileges in large systems. On the one hand every user is mapped to a set of roles. On the other hand every role is mapped to a set of permissions. This concept reduces redundancy in systems with large or fast growing sets of subjects and objects and also models real scenarios closer to reality (e.g. job function in an organization).

Definition 2.2 (RBAC).
A RBAC$_0$ model is a tuple $(U, R, P, S, UA, PA, user, roles)$ with

- U as set of *users*
- R as set of *roles*
- $P = 2^{O \times OP}$ as set of *permissions*
    - O is the set of *objects*
    - OP the set of *operations*)
- S as set of *sessions*
- $UA \subseteq U \times R$ as a many-to-many *user-to-role assignment relation*
- $PA \subseteq P \times R$ as a many-to-many *permission-to-role assignment relation*
- $user : S \rightarrow U$ as function, that maps every session to a single user
- $roles : S \rightarrow 2^R$ as function, that maps every session to set of roles

RBAC is called a family of security models, because there is a base model (RBAC$_0$) and three models that extend this base model with advanced concepts. RBAC$_1$ extends RBAC$_0$ with a role hierarchy, because in reality it turned out that roles are often combinations of other already existing roles. This reduces redundancy and improves its scaling behavior with increasing number of users and objects even further. The role hierarchy is defined as partial order $RH \subseteq R \times R$ on R (also written as $\geq$).

RBAC$_2$ extends RBAC$_0$ with a set of constraints C to restrict values of the model components $UA, PA, user, role$ or RH (e.g. forbid the activation of two certain roles at the same time). Constraints can be highly individual and specific to a certain application. However, Sandhu et al. [1996] mentions three main concepts for constraints, that seem to be used frequently and are therefore reasonable to implement: *Mutually exclusive roles*, *cardinality* and *prerequisite roles*.

RBAC$_3$ combines the concepts of RBAC$_1$ and RBAC$_2$.

2.1.3 RBAC notation by Sandhu at al.

Sandhu et al. [1996] used a graphical notation in his work to visualize the RBAC model family with all its elements and relationships (see figure 2.1). The RBAC model is a well-established security models among many others. Also regarding its relevance in literature – especially for its contribution to the standardized NIST RBAC model [Computer Security Resource Center, 2016] – we want to take a look at its graphical notation.

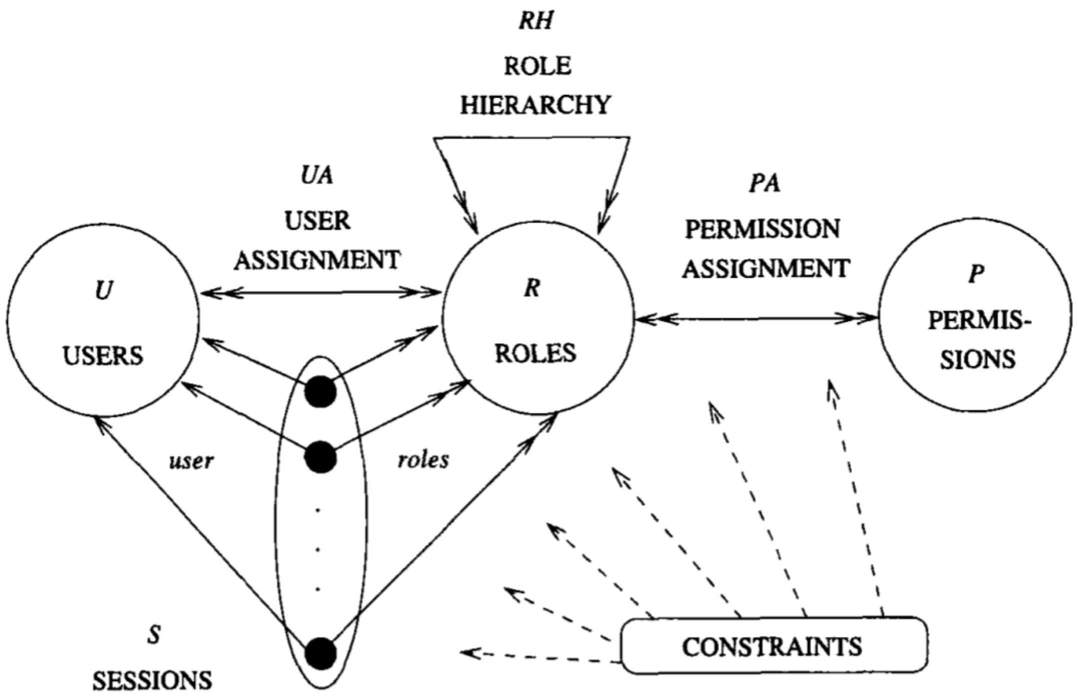

Figure 2.1: Graphical notation used by Sandhu et al. [1996] to visualize $RBAC_3$

Unfortunatly Sandhu et al. [1996] gives no details on how or why the notation is build up the way it is in his work. Following graphical elements are used for this notation:

- A circle or ellipse represents a set (U, R, P, S).
- A rounded rectangle represents a special set – the set of constraints $(Constraints)$.

- A normal-headed unidirectional arrow represents a one-to-one assignment ($user$).
- A double-headed unidirectional arrow represents a one-to-many assignment ($roles$).
- A double-headed bidirectional arrow represents a many-to-many assignment relation (UA, PA, RH).
- A normal-headed unidirectional dotted arrow represents a constraint. Six arrows of this type are pointing symbolically from the set of constraints in the direction of all model components constraints can be potentially defined on.

A set has its name and its identifier in its center. An arrow – except the constraint arrow – has its name and its identifier next to it.

Furthermore the *Sessions* set (S) is not directly connected with arrows to other components. It is drawn with large black dots inside, which represent single elements of this set. These set elements are then connected to other components with certain arrows. Three small black dots (in this case aligned horizontally) indicate that there are potentially more than just those three set elements (the number of sessions is not limited per definition).

This diagram is often used in a simplified form (for example in [Amthor, 2018]) that directly connects the *Sessions* set – like the other sets – with just one arrow to *Users* and one to *Roles* and does not display single set elements of *Sessions*. Also worth mentioning is that RH is drawn as a loop. It is pointing from *Roles* (R) to *Roles* with a double-headed bidirectional arrow.

2.2 Aspect-oriented Security Engineering and Entity-Labeling Aspect

In this section the Aspect-oriented Security Engineering proposed by [Amthor, 2018] is described with focus on its Entity-Labeling aspect. The content of this section will act as fundamental basis of information for designing a graphical specification language for the Entity-Labeling aspect later in chapter 3.

2.2.1 Aspect-oriented Security Engineering

Aspect-oriented Security Engineering (AOSE) is a new approach proposed by Amthor [2018] to improve the process of model-based security engineering. With an increasing number and complexity of security models the engineering process that goes along also became more complex and therefore prone to human errors.

Amthor [2018] claims to close semantic gaps between the steps model engineering, model analysis and formal policy specification in the process of security engineering to reduce the potential impact of human errors. These steps are depicted in figure **??**.

For this AOSE roughly adopts the idea of the aspect-oriented programming paradigm. It tailors all steps to aspects, which are non-functional requirements of the engineering process like determining requirements of policy semantics or analyzing certain security goals. There are two major classes Amthor [2018] proposes for aspects regarding AOSE: related to policy semantics and related to policy analysis. One of several aspects proposed is Entity-Labeling.

According to Amthor [2018] an aspect-oriented security model is formally defined as:

Definition 2.3 (Aspect-oriented security model).
A aspect oriented security model is defined as $\langle \mathcal{M}, \mathcal{A}, sem \rangle$ with:

- $\mathcal{M}$ as finite set of model component identifiers
- $\mathcal{A}$ as set of semantic category identifiers (aspect)
- $sem : \mathcal{A} \to 2^{\mathcal{M}}$ as semantical application of $\mathcal{A}$ to $\mathcal{M}$

2.2.2 Entity-Labeling Aspect

The Entity Labeling Aspect (EL) is one possible aspect in context of AOSE. It is designed to formally specify policy semantics typically found in operating systems and middleware systems. It bridges the gap between informal policy and formal model by supporting its transformation. EL classifies model components into six semantic categories:

1) Entity Set (ES)
A set of entity identifiers in the domain of an AC system.

2) Label Set (LS)
A set of legal label values.

3) Label Assignment (LA)
An association between entities and labels

4) Access Rule (AR)
A logical rule that defines, based on a set of entity labels, which operations may be legally performed on entities corresponding to these labels. Model components in AR thus reflect a policy's access control function (ACF).

5) Relabeling Rule (RR)
A logical rule for legal label changes

6) Model Constraint (MC)
Constraints over model components that must be satisfied in every model state

According to these categories, we define the EL aspect:

Definition 2.4 (Entity-Labeling aspect).
The entity labeling aspect of a security policy is defined as $\mathcal{A}_{EL} = \{ES, LS, LA, AR, RR, MC\}$ where ES, LS, LA, AR, RR, and MC denote categories of model components with the semantics defined in Defs. 4.2–4.7.

2.3 Graphical notation models

2.3.1 Unified Modeling Language

The *Unified Modeling Language* (UML) is a graphical modeling language to specify, visualize, design and document software [The Object Management Group, 2015]. UML became a widely used tool in software development and was standardized by The Object Management Group in 1997 and also was approved as official ISO standard in 2005 [International Organization for Standardization, 2005].

UML defines terms and relationships between those terms, as well as a graphical notation for those. The current version of UML (2.5) contains 14 different types of diagrams. Every diagram illustrates information according to one specific aspect. UML diagrams are classified into static and dynamic types, but they can have fuzzy borders and mixed forms as well. Static diagram types describe structural properties, dynamic diagram types behavioral properties.

Components an UML diagram is build up with are classified into *elements* and *relationships*:

1) Elements

Elements are the main objects in a UML diagram. Graphically elements are based on closed shapes like boxes or ellipses. They usually have a name or identifier in form of text in the center of their shape. Horizontal lines can be used to separate an element into two or more sections (see *class diagram*). Examples: Classes, components, nodes, objects, packages (see figure 2.2)

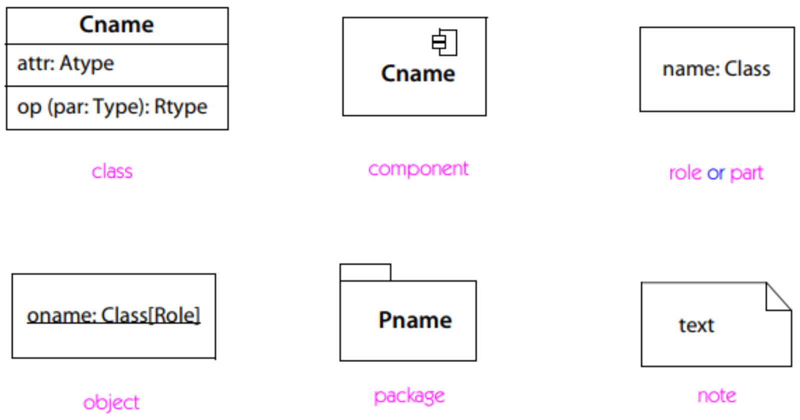

Figure 2.2: Selection of UML elements

2) Relationships

Relationships can connect two or more elements. Graphically relationships are based on lines. Optionally they can have a shape at one end to indicate some kind of direction. They can vary in form by having differently drawn lines (solid, dotted), shaped arrows. They can have several angles (preferable 90° or 45°) to run on a clearly laid out path. Lines do not collide if possible. Examples: Dependency, association, generalization, include, extend (see figure 2.3)

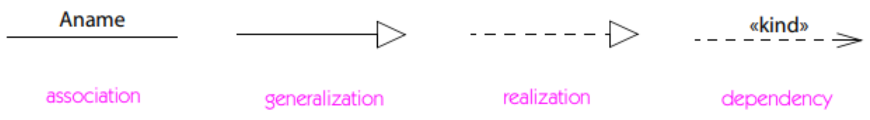

Figure 2.3: Selection of UML relationships

Class diagrams can have different levels of detail. For example the class element has a simple symbolic form as well as an extended form with detailed information about its key words, attributes and operations (see figure 2.4).

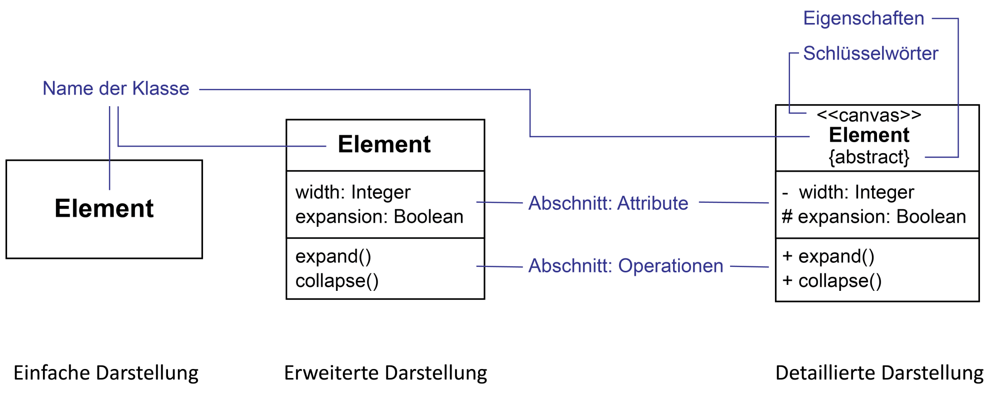

Figure 2.4: Class diagrams with different level of detail

UML has symbols and textual identifiers to visualize certain properties. Class diagrams for example have the following symbols to indicate visibility or attributes and operations: + for *public*, # for *protected* and - for *private*. With *multiplicity* an element can be constrained regarding kind and number of its values. The format for its bounds in Backus-Naur form (BNF) is defined by the UML specification 2.5 [The Object Management Group, 2015] as in listing 2.1.

```
⟨multiplicity⟩              ::= <multiplicity-range> [ [ ‘{’ <order-designator> [ ‘,’
                                <uniqueness-designator> ] ‘}’ ] | [ ‘{’ <uniqueness-
                                designator> [ ‘,’ <order-designator> ] ‘}’ ] ]

⟨multiplicity-range⟩        ::= [ <lower> ‘..’ ] <upper>

⟨lower⟩                     ::= <value-specification>

⟨upper⟩                     ::= <value-specification>

⟨order-designator⟩          ::= ‘ordered’ | ‘unordered’

⟨uniqueness-designator⟩     ::= ‘unique’ | ‘nonunique’
```

Listing 2.1: UML multiplicity syntax in BNF

For the sake of convenience we assume the literal <value-specification> being an integer number or * for infinite.

Furthermore it is worth mentioning that UML does not use any color for its diagrams. This makes it compatible, easy and fast for hand-drawing. Coloring always remains a not officially specified option to highlight certain parts of a diagram, e.g. to set a focus for an accompanying text.

2.3.2 Entity-Relationship model

The entity-relationship model (ER) is a model to describe classified objects (entities) and how they are interrelated. It was developed to illustrate information in databases on a conceptual level and became a popular tool for requirement analysis in this domain. In contrast to UML (see 2.3.1), which is a collection of multiple diagram types, ER only has one type of diagram: The Entity-Relationship diagram (ERD).

Also unlike UML ER is not an officially specified standard. Furthermore ER has some slightly different notation variants (Chen, Bachman, etc.), which will be discussed in section 2.3.2.1. The figures in this subsection are in Chen notation. Figure 2.5 shows an example of an ERD.

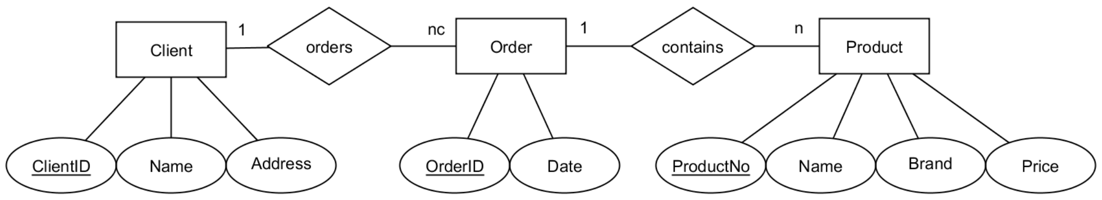

Figure 2.5: ERD example

ERDs are build up with the following elements [Kleuker, 2011]:

1) Entity type

The most important artifact of ER is probably the entity. It represents an individual and unambiguously identifiable object. An entity is characterized by its properties. An *entity type* is a template for entities that summarizes all their attributes. Entities can be interpreted as instances of an entity type with concrete values. Graphically an entity type is a simple rectangle with its name in its center.

Figure 2.6: Entity object of an ERD

2) Relationships

Relationships can connect two or more entities and describe how they are interrelated. In contrast to UML (see 2.3.1) relationship is not represented with a simple lines, but with a rhombus with the relationship name in its center. This rhombus is then connected to two or more entities with simple lines. These lines may have symbols at their end to indicate properties of the connected entities in context of the relationship.

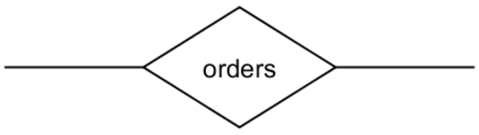

Figure 2.7: Relationship object of an ERD

3) Attributes

Attributes are properties of entites or relationships. Graphically an attribute is a single ellipse with its name in its center. It is connected to an entity or to a relationship with the help of a simple line.

Description

Figure 2.8: Attribute object of an ERD

To specify the amount of entities in context of a relationship the following indicators for *cardinalities* are used:

- **1** to indicate a relationship to exactly one entity.
- **c** to indicate a relationship to no or one entities.
- **n** to indicate a relationship to one or multiple entities.
- **nc** to indicate a relationship to no, one or multiple entities.

Cardinalities can be upper case (N) or lower case (n) with no semantic difference. A relationship can be read in two directions. For example the *orders*-relationship in figure 2.5:

- An order is connected to exactly one client (left-to-right).
- A client is connceted to no, one or multiple orders (right-to-left).

An entity type needs to have a *primary key* defined in order to have its elements unambiguously identifiable. The primary key is an attribute or a combination of attributes of the entity type and has to be unique among all other attribute values. A primary key attribute is graphically indicated with an underlined name (see *ClientID* in figure 2.5).

Besides this basic system, ERDs can indicate some additional information. For example besides those elements described above, there are variants of them with a double frame line to indicate a *weak* property (no primary key, therefore dependent to parent elements) or a dashed frame line to indicate a *multivalued* property. In context of this work those will not be relevant.

Notable is that an ER model can be visualized with the help of UML to some degree. When entity types are treated as classes, entities as objects, relationships as associations, attributes as instance variables and cardinalities as multiplicities an UML *class diagram* can be used to build up a diagram similar to an ERD.

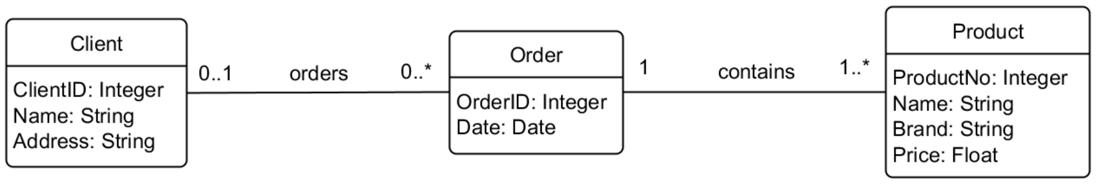

Figure 2.9: Information from figure 2.5 modeled as UML class diagram

As seen in figure 2.9 the diagram differs from the diagram in figure 2.5 in almost all aspects, but still they are easy to read and to translate into each other.

2.3.2.1 Notations

As already mentioned ER is not standardized and has different notation variants. The notation variants only differ in their symbol usage at the end of lines. Here is a short overview on how some of these differ in their appearance:

2.4 Gestalt laws and human optical perception

2.4.1 Gestalt laws

2.4.2 Human optical perception

2.5 GUI Design

2.5.1 Design Patterns

2.5.2 Usability

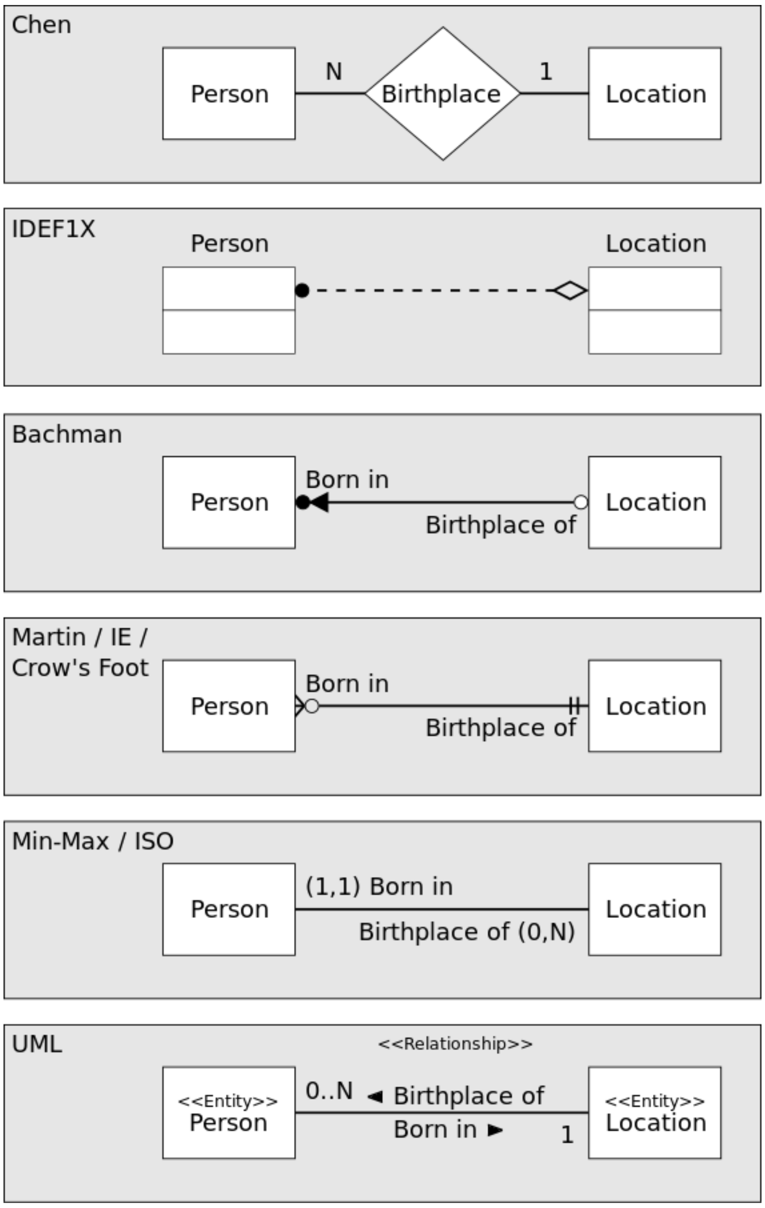

Figure 2.10: Simple example in different ER notation variants

CHAPTER 3

Design: Graphical specification language

3.1 Concept

approach, basic ideas, adoptions from literature

3.2 Elements

3.3 Relationships

3.4 Structure

3.5 Visualization on the higher abstraction level

CHAPTER 4

Design: Editor GUI

4.1 Structure

4.2 Sections

CHAPTER 5

Implementation

5.1 Implementation base

Qt, MVC

5.2 Structure

5.3 GUI sections

CHAPTER 6
Evaluation

6.1 Graphical specification language

6.2 Editor

CHAPTER 7
Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

CHAPTER 8 Summary

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Bibliography

Peter Amthor. *An Aspect-oriented Approach to Model-based Security Engineering.* dissertation, Technische Universität Ilmenau, 2018.

Computer Security Resource Center. Role based access control. `https://csrc.nist.gov/Projects/Role-Based-Access-Control`, 2016. [Online; accessed June 13, 2018].

International Organization for Standardization. Iso/iec 19501:2005 (omg-uml ver 1.3). `https://www.iso.org/standard/32620.html`, 2005. [Online; accessed 13-June-2018].

Stephan Kleuker. *Grundkurs Datenbankentwicklung.* Vieweg+Teubner Verlag, 2011. ISBN 978-3-8348-1481-4.

Butler W. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8(1):18–24, January 1974. ISSN 0163-5980. doi: 10.1145/775265.775268. URL `http://doi.acm.org/10.1145/775265.775268`.

Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, pages 38–47, 1996.

The Object Management Group. Unified modeling language specification version 2.5. `https://www.omg.org/spec/UML/2.5/`, 2015. [Online; accessed 07-June-2018].