

DISSERTATION

A Formal Treatment of UML Class Diagrams as an Efficient Method for Configuration Management

AUSGEFÜHRT ZUM ZWECKE DER ERLANGUNG DES AKADEMISCHEN GRADES EINES
DOKTORS DER TECHNISCHEN WISSENSCHAFTEN UNTER DER LEITUNG VON

A.O.UNIV.PROF. DI. DR. GERNOT SALZER
ARBEITSBEREICH FÜR THEORETISCHE INFORMATIK UND LOGIK
INSTITUT FÜR COMPUTERSPRACHEN (E185)

EINGEREICHT AN DER FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT WIEN VON

MAG. DI. **Ingo Feinerer** BAKK.

MATRIKELNUMMER 0125130
FELIXDORFER GASSE 11
2700 WIENER NEUSTADT
ÖSTERREICH

WIEN, MÄRZ 2007

Kurzfassung

Eine Konfiguration beschreibt die Anordnung funktionaler Einheiten nach deren Natur, Anzahl und Schlüsseigenschaften. Dabei sind unter funktionalen Einheiten Software- oder Hardwarekomponenten wie Computerprogramme, elektronische Schaltkreise, oder Module eines Rechners zu verstehen. Der Begriff „Configuration Management“ bezeichnet dabei die Spezifikation von zulässigen Anordnungen in für den Benutzer möglichst einfacher Form und die Anordnung der spezifizierten Elemente nach bestimmten Optimalitätskriterien. Dazu gehören etwa die Erfüllbarkeit von Spezifikationen oder die Minimalität der berechneten Konfigurationen. Die Überprüfung und Berechnung von Konfigurationen erfordert effiziente Methoden, da die Bearbeitung der immer größer werdenden Konfigurationen zunehmend in Echtzeit erfolgen soll.

Die „Unified Modeling Language“ ist ein industrieweit anerkannter Standard geworden und stellt Sprachelemente zur Verfügung, wie sie in realen Anwendungen des „Configuration Management“ benötigt werden. Hinzu kommt, dass sehr viele Software Entwickler bereits mit UML vertraut sind. Deshalb verwenden wir UML als Basis für unsere Betrachtungen, wozu wir für UML eine formale Semantik entwickeln und darauf aufbauend die Begriffe der Konsistenz und Minimalität definieren. Es werden Methoden zur effizienten Berechnung zur Konsistenzprüfung von UML Klassendiagrammen als auch zum Finden von minimalen Lösungen von UML Konfigurationen vorgestellt. Dabei werden sowohl binäre Verbindungstypen als auch Verbindungstypen höherer Stelligkeit betrachtet, wobei letztere aufgrund der speziellen UML Semantik eine Sonderbehandlung brauchen. Zusätzlich wird die inkrementelle Erweiterbarkeit von UML Konfigurationen andiskutiert.

DISSERTATION

**A Formal Treatment of UML Class Diagrams
as an Efficient Method
for Configuration Management**

Ingo Feinerer

THEORY AND LOGIC GROUP
INSTITUTE OF COMPUTER LANGUAGES
VIENNA UNIVERSITY OF TECHNOLOGY

ADVISOR: GERNOT SALZER

VIENNA, MARCH 2007

Abstract

The concept of a configuration describes the arrangement of functional units according to their nature, number, and chief characteristics. Functional units may be software or hardware components like computer programs, electronic circuits, or parts of a machine. Configuration management is concerned with the specification of admissible arrangements in a natural way and with setting them up according to certain criteria of optimality. Typical problems to solve are the satisfiability of specifications and the minimality of computed configurations. The steady increase in the size of specifications and the demand for real-time computations require efficient methods to attack these problems.

The Unified Modeling Language has become a widely accepted standard in industry and offers features capable of modelling real-world situations in configuration management. Many (software) engineers are already acquainted with UML. Therefore we use UML as basis for our considerations and define a formal semantics and the notions of consistency and minimality. We present efficient methods for checking the consistency of UML class diagram specifications and for finding minimal solutions of UML configurations. We discuss both binary association types with uniqueness constraints and association types of higher arity which need special treatment due to the special UML semantics. We also discuss the problem of updating UML configurations incrementally.

Acknowledgements

Especially I would like to thank my family for their organisational and financial support throughout the years.

I am grateful to Andreas Falkner and Gottfried Schenner for introducing me to the area of configuration management at Siemens Austria.

Additionally I appreciate the feedback of several anonymous reviewers at the 11th International Workshop on Formal Methods for Industrial Critical Systems and the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. It helped me a lot to improve the results in this thesis.

Special thanks need to be dedicated to my advisor Gernot Salzer. He introduced me to the realm of scientific research. He invested an enormous amount of time in discussions with me and supported me actively in writing scientific papers. Often he came up with interesting ideas which showed to be very useful in my research. Gernot was always interested in fostering my scientific activities. Additionally I could profit a lot from his broad \LaTeX knowledge. Summarising, his input was invaluable during the last years. Thank you very much, Gernot!

Contents

1. Introduction	10
2. ER Diagrams and Their Satisfiability	12
2.1. Entity Relationship Diagrams	12
2.2. Constraint Satisfiability in ER Diagrams	14
2.3. Minimal SERM Solutions	17
3. UML Class Diagrams	19
3.1. Unified Modeling Language	19
3.2. Semantics of Multiary Associations	22
3.2.1. Reduction of Multiary to Binary Associations	24
3.2.2. Semantics of Minimum Multiplicity in Ternary UML Associations	26
4. A Formal Semantics for UML Class Diagrams	29
4.1. Configurations and Specifications	29
5. UML Class Specifications with Binary Associations	34
5.1. From Specifications to Linear Diophantine Inequations	34
5.2. Solving Linear Diophantine Inequations	39
6. UML Class Specifications with Multiary Associations	56
6.1. Extended Linear Diophantine Inequations	56
6.2. Solving Extended Linear Diophantine Inequations	66
7. Incremental Update of UML Class Configurations	68
7.1. Different Requirements for Changing Configurations	68
7.2. Strategies for Handling Incremental Updates	69
8. Conclusion	72
A. Resources	73

List of Algorithms

- 1. Constructing the links of a configuration 40
- 2. Compute paths with maximal weights 43
- 3. Compute a minimal rational solution 47
- 4. Compute a minimal integer solution 52

- 5. Constructing the links of a configuration with multiary associations 63

List of Examples

4.	Inconsistent UML specification	32
5.	γ and δ conditions for an asymmetric specification	32
7.	Problems of uniform view on asymmetric specifications	33
9.	Satisfiability conditions for a symmetric specification	35
11.	Satisfiability conditions for an asymmetric specification	36
15.	UML-constraints	41
21.	Consistency check for a specification with three classes	44
23.	Computing the minimal rational solution	46
25.	Counter-example for the general integer minimality of the lcm-approach	50
27.	Computing the minimal integer solution	51
29.	Discussion on the complexity of the minimal integer solution algorithm	53
30.	Constructing links for a configuration	54
34.	An enumeration satisfying Conjecture 33	60
35.	Sequences with pairwise coprime domains satisfy Conjecture 33	60
36.	A sequence satisfying Conjecture 33	60
38.	Satisfiability conditions for a symmetric multiary specification	62
39.	Excursion to asymmetric multiary associations	64
40.	Checking Diophantine inequations in the multiary case	67
41.	Extending configurations versus extending inequations	70

List of Figures

2.1.	An ER diagram modelling a relationship between employees and projects	13
2.2.	A SERM schema which is not strongly satisfiable	14
2.3.	The weighted directed graph to Figure 2.2	16
3.1.	A hierarchy for UML 2.0 diagram types	20
3.2.	Example of a UML class diagram	22
3.3.	A 4-ary association	23
3.4.	A ternary association interpreted under look-here semantics	24
3.5.	Decomposition of Figure 3.4 into binary associations via mediator class	25
3.6.	Decomposition into pairwise binary associations	25
3.7.	A ternary UML association with generic minimal multiplicities	26
4.1.	UML specification that is weakly but not strongly consistent	32
5.1.	UML specification for Example 9	35
5.2.	UML specification (uniq constraints omitted)	48
5.3.	UML class diagram leading to long cycles for Algorithm 4	53
6.1.	A UML specification with a generic ternary association	57
6.2.	A 4-ary association (without drawn uniq constraints)	64
6.3.	A UML specification with an asymmetric ternary association	65
7.1.	A UML specification in the context of extending configurations	70

1. Introduction

*Quantum Mechanics is a lovely introduction to Hilbert Spaces!
Overheard at last year's Archimedean's Garden Party*

The *Unified Modelling Language* (UML) [Object Management Group, 2005] has gained wide acceptance in software engineering as a universal formalism for object-oriented modelling, offering notations for describing class relationships, component systems, processes, use cases, and more. Additional constraints can easily be imposed with the help of the *Object Constraint Language* (OCL) [Object Management Group, 2006]. In the past decade UML and OCL have been also considered for specifying configurations [Felfernig et al., 2002b,a].

The term *configuration* as used in this thesis refers to an arrangement of functional units according to their nature, number, and chief characteristics in accordance to definitions given by The Alliance for Telecommunications Industry Solutions [2000]. Functional units may be software or hardware components like computer programs, electronic circuits, or parts of a machine. A major issue is to specify admissible arrangements in a natural way and to set them up according to certain criteria of optimality. These activities are called *configuration management*.

UML class diagrams offer *multiplicities* to restrict the number of relations between objects. This formalism is already expressive enough to specify relevant configuration problems like railway interlocking systems [Schenner and Falkner, 2002]. Compared to logic-oriented approaches, UML diagrams have the advantage that (software) engineers are acquainted with the formalism and that many tools exist for composing and manipulating UML specifications.

Using UML for configuration management brings about some problems that usually do not bother software engineers in object-oriented modelling. Most notably, the extensive use of multiplicities may lead to inconsistent diagrams that admit only the trivial configuration (no objects per class). Therefore algorithms are needed for detecting inconsistencies. Moreover, one usually wants to find small (or even minimal) configurations satisfying the specification. In short, configuration management requires *formal reasoning* about configurations.

One approach is to translate UML diagrams to some logic. E.g., description logic (DL) leads to concise specifications with a clear, mathematically defined semantics. Berardi et al. [2001, 2005] show how a wide range of UML/OCL elements can be translated to DL formulas. Other approaches (see e.g. Schenner and Fleischanderl [2003]) use sub-classes of first-order logic. These logics are well-developed and one can build on the results of many decades of research. Furthermore, they are very expressive and can easily

integrate additional information from other sources within a uniform framework. On the negative side, the expressiveness leads to a high complexity of the reasoning tasks (NP-hard or worse). E.g., consistency checking using the description logic of Berardi et al. [2005], called *ALUQI*, is EXPTIME-complete [Baader et al., 2003].

Calvanese and Lenzerini [1994] investigate is-a and cardinality ratio constraints in *Entity Relationship* (ER) schemata. They solve the satisfiability problem with disequations and so-called expansion mechanisms, and show that this task is decidable in EXPTIME. Calvanese et al. [1994] discuss a framework for class-based representation formalisms like ER diagrams. They use the description logic *ALUNTI* as representation language and show how to integrate several formalisms into this framework. They do not cover UML diagrams, but tackle the problem of finite model reasoning by disequations and expansions. Their formalism is expressive, but leads to EXPTIME completeness.

In this thesis we choose a different approach. We encode UML class diagrams as inequations over non-negative integers and show that consistency can be checked in polynomial time. Moreover, we give an algorithm to compute minimal configurations. Using inequations is not entirely new: Lenzerini and Nobili [1990] use them to check the satisfiability (consistency) of dependency constraints in ER schemata. Engel and Hartmann [1995] compute minimal solutions for semantic ER schemata with the Ford-Bellman algorithm. Since UML class diagrams are derived from ER schemata, these results can be reused: Binary UML-associations marked with the attribute ‘unique’ correspond to binary ER-relations. Differences surface when it comes to non-unique and n -ary associations. Associations with the attribute ‘non-unique’ were not considered by the ER community but are part of the UML standard. This new attribute requires special treatment, in particular when computing admissible configurations. Concerning n -ary associations, ER schemata offer two semantics: the *look-across approach* going back to Chen [1976] and the *look-here approach* introduced with Merise [Rochfeld, 1986]. Lenzerini and Nobili adhere to the latter view, while UML prescribes the former.

The novel contributions in this thesis are the definition of a particular formal semantics for UML class diagrams with multiplicities and uniqueness constraints, the transformation of UML class diagrams to systems of Diophantine inequations and their efficient solution such that properties like satisfiability of specifications and minimality of configurations can be easily computed, the generalisation of our approach to n -ary associations under the look-across semantics, and a discussion on updating configurations incrementally.

Chapter 2 reviews existing approaches in the ER literature followed by an overview of UML class diagrams in Chapter 3. In Chapter 4 we define a formal semantics for UML class diagrams which we will use as basis for our investigations regarding consistency and minimality. We discuss both properties and present efficient algorithms for binary (Chapter 5) and n -ary associations (Chapter 6). Finally, Chapter 7 gives an outlook on incremental updates of configurations.

2. ER Diagrams and Their Satisfiability

Computer, n.:
An electronic entity which performs sequences of useful steps in a totally understandable, rigorously logical manner. If you believe this, see me about a bridge I have for sale in Manhattan.

In this chapter we review the ER literature related to our work. We start by describing ER diagrams and give a formalisation of them. Further we investigate strategies for checking their consistency and for computing minimal elements in the ER context.

2.1. Entity Relationship Diagrams

One of the most used diagram types in computer science are *Entity Relationship Diagrams* [Chen, 1976] (ER diagrams). ER diagrams use the concepts of entities, relationships, roles, and mappings.

Entities: an (ER-)entity captures the type of objects to be modelled. In modern terminology (like in object oriented programming languages, or UML) entities would be called classes. Entities are visualised by rectangular boxes. Instances of entities are called objects.

Relationships model associations or connections between two or more entities, i.e., n -ary associations can be specified in ER. Relationships are visualised by diamond-shaped boxes. An instance of a relationship is called relation.

Roles name parts of relationships to identify ends. Roles are often neglected if the situation is clear from context.

Mappings impose constraints on the relations between objects. Chen distinguishes $1 : N$, $M : N$ and $1 : 1$ mappings corresponding to one-many, many-many, and one-one relations between objects. Originally, M and N did not specify concrete values but only indicated relationship types. In the following development of ER diagrams, generalisations interpreted $M, N \in \mathbb{N}$ and introduced ranges, like, $1 : M..N$.

Chen proposed to use a *look-across* style for how to read relationships and mappings. This means for an n -ary association that $n - 1$ entities are instantiated. This set of instantiated relationships is now restricted by the relationship's stated number at the side of the remaining object. Thus in the binary case one can read the number of allowed

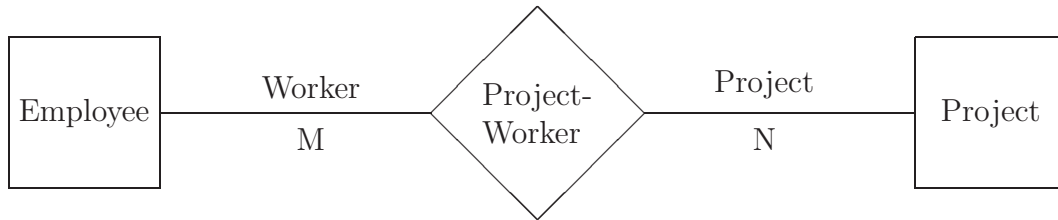


Figure 2.1.: An ER diagram modelling a relationship between employees and projects

partner objects for a single object by looking across the relationship. For the n -ary case this definition leads to different interpretations—we will discuss this issue in detail in Section 3.2.

Figure 2.1 depicts an example entity relationship diagram (modelled after an example of Chen [1976, page 19]). The two entities `Employee` and `Project` are drawn within rectangular boxes, whereas the relationship `Project-Worker` is drawn within a diamond box. `Worker` and `Project` identify roles, `M` and `N` define a $M : N$ mapping.

A Formal Definition of ER Diagrams

Lenzerini and Nobili [1990] present a formal definition for ER diagrams which is useful in many ways. First, it unveils a way of formalising modelling instruments, second it proves useful when we talk about certain properties (like satisfiability) in an exact manner, and third, it shows us the differences between this and our formalisation of UML which will be found later in this thesis.

They use the (natural language) definitions as already given in this section as basis but go into more detail for specific notions: a relationship instance r , written as

$$r = \{ \langle e_1, U_1 \rangle, \langle e_2, U_2 \rangle, \dots, \langle e_n, U_n \rangle \} ,$$

connects n element instances e_1, \dots, e_n through roles U_1, \dots, U_n .

A *semantic entity-relationship model* (SERM) schema consists of a set of entities, a set of relationships, a set of roles and a set of cardinality ratio constraints. An instance of a SERM schema are objects and relations satisfying following constraints:

- the relations are well-typed, i.e., each instance of a relationship has to contain the roles of the relationship with objects instantiating the corresponding entities,
- the roles are unique, and
- relations are uniquely characterised by the sets of their role-object pairs, i.e., instances with the same set are identical.

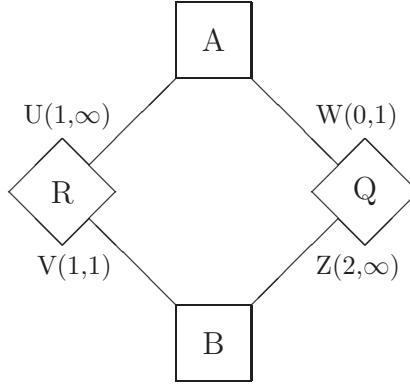


Figure 2.2.: A SERM schema which is not strongly satisfiable [Lenzerini and Nobili, 1990, page 456]

2.2. Constraint Satisfiability in ER Diagrams

Since ER diagrams share several properties with UML diagrams by their definition, consistency results for ER constraints are of interest to our work.

A major contribution in the analysis of consistency and satisfiability of constraints in entity relationship schemata goes back to Lenzerini and Nobili [1990]. They investigate the satisfiability of so-called dependency constraints in ER SERM schemata.

Let R be an ER relationship and let E be an ER entity. A *cardinality ratio constraint* is of the form

$$E(U) \xrightarrow{(M,N)} R .$$

It expresses that entity E must be connected to relationship R via role U with at least M ($M \geq 0$) links and at most N ($M \leq N \leq \infty$) links.

Note that this definition corresponds to the look-here semantics for associations (see the discussion in Section 3.2). This implies that certain results by Lenzerini and Nobili [1990] cannot be directly used in the UML context due to its look-across association semantics (confer its definition in Section 3.1). This applies especially to n -ary associations, $n \geq 3$, where look-here and look-across semantics significantly differ.

Besides considering classical satisfiability—a schema is said to be satisfiable if some instance of a schema exists which satisfies all integrity constraints, i.e., inherent constraints as mentioned in the previous section and explicit cardinality ratio constraints—they also cope with strong satisfiability. This is especially useful in class based modelling mechanisms since strong satisfiability requires for each class C in an ER schema S the existence of an instance of S such that the set of instances for C is not empty. Strong satisfiability forbids classes which never get instantiated (and hence are useless in many scenarios).

The satisfiability check is based on a transformation of the ER system with its cardinality ratio constraints to a system of inequations with a variable \hat{E} for each entity E and a variable \hat{R} for each relationship R . The system of inequations is as follows

[Lenzerini and Nobili, 1990, page 456]:

$$\begin{aligned}\hat{R} &\geq M \cdot \hat{E} \\ \hat{R} &\leq N \cdot \hat{E}\end{aligned}$$

for each cardinality ratio constraint $E(U) \xrightarrow{(M,N)} R$,

$$\hat{E} > 0$$

for each entity E , and

$$\hat{R} > 0$$

for each relationship R .

Each solution of the system of inequations corresponds to an instance of the ER diagram: \hat{E} gives the number of objects and \hat{R} gives the number of relations. The proof gives an algorithm for connecting objects with relations.

A further important aspect in their paper is a method for analysing unsatisfiable schemata. The method is based on the construction of a weighted directed multigraph (V, A) , where

- the vertices V match the set of classes, and
- the arcs A are built by introducing an arc from the node corresponding to E to the node corresponding to R labelled with N , and an arc from the node corresponding to R to the node corresponding to E labelled with $1/M$ (∞ if $M = 0$), for each connection in the schema between E and R with its cardinality ratio constraint $E(U) \xrightarrow{(M,N)} R$.

Since multiplying the values on a path π in this graph simulates the transitive closure of cardinality ratio constraints, a necessary and sufficient criterion for identifying inconsistent schemata is the non-existence of a so-called correct assignment for each cycle in the graph. An assignment φ , mapping nodes to positive real numbers, is correct if and only if

$$\frac{\varphi(n_f)}{\varphi(n_s)} \leq \prod_{a \in \pi} l(a)$$

holds for each arc in the graph, where a denotes the arcs in the path π with starting node n_s and final node n_f , and $l(a)$ denotes the label of arc a .

Finally, Figure 2.2 shows us a SERM schema as originally found in the discussed paper of Lenzerini and Nobili [1990]. A and B drawn in boxes denote entities, R and Q drawn in diamonds denote relationships, respectively. U, V, W , and Z denote roles with mappings in parentheses. We have the cardinality ratio constraints

$$\begin{array}{ll} A(U) \xrightarrow{(1,\infty)} R & B(V) \xrightarrow{(1,1)} R \\ A(Q) \xrightarrow{(0,1)} Q & B(Z) \xrightarrow{(2,\infty)} Q \end{array} .$$

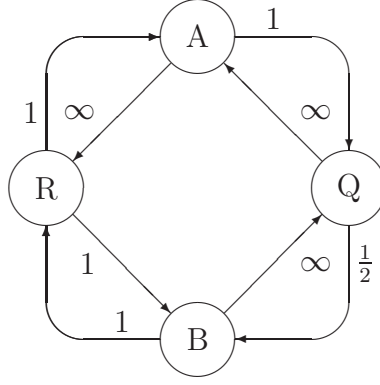


Figure 2.3.: The weighted directed graph to Figure 2.2

We immediately see that the schema is not strongly satisfiable since one A needs exactly one B enforced by role R but role Q prescribes at least 2 B s for the entity A . Formally we check the satisfiability by generating following inequations

$$\begin{array}{ll} A > 0 & R > 0 \\ B > 0 & Q > 0 \end{array}$$

and

$$\begin{array}{ll} \hat{R} \geq 1 \cdot \hat{A} & \hat{R} \leq \infty \cdot \hat{A} \\ \hat{R} \geq 1 \cdot \hat{B} & \hat{R} \leq 1 \cdot \hat{B} \\ \hat{Q} \geq 0 \cdot \hat{A} & \hat{Q} \leq 1 \cdot \hat{A} \\ \hat{Q} \geq 2 \cdot \hat{B} & \hat{Q} \leq \infty \cdot \hat{B} . \end{array}$$

Within this system of inequations we can derive $2 \cdot \hat{B} \leq \hat{B}$ with $\hat{B} > 0$. This proves our initial claim that the schema is not strongly satisfiable.

Figure 2.3 visualises the weighted digraph belonging to the SERM schema as constructed by the above algorithm, i.e., vertices are introduced for the set of entities and relationships, and the arcs are labelled as described above. Since the corresponding schema is not strongly satisfiable we cannot find a correct assignment for each path in the digraph. W.l.o.g., assume $A = 1$. This enforces $R = 1$ and $B = 1$ on the path $A \rightarrow R \rightarrow B$ as otherwise it would be an incorrect assignment. Simultaneously, $A = 1$ only permits $Q = 1$ and $B = 2$ to be a correct assignment on the path $A \rightarrow Q \rightarrow B$. The two assignments are incompatible, i.e., we see that the existence of an incorrect assignment shows up for graphs representing not strongly satisfiable SERM schemata.

2.3. Minimal SERM Solutions

Based on the work of [Lenzerini and Nobili \[1990\]](#), [Engel and Hartmann \[1995\]](#) present an algorithm for constructing minimal instances for given SERM schemata (also called SER scheme).

They start by assigning a bipartite digraph to a given SER scheme. This allows them to use results from graph theory directly which will show to be very useful in their investigations. Formally:

A *SER scheme* is a bipartite graph $G = (V, A)$ such that

$$\begin{aligned} V &= E \cup R \\ A &\subseteq E \times R \end{aligned}$$

with the functions

$$\begin{aligned} \alpha_S: A &\mapsto \mathbb{N} \cup 0 \\ \beta_S: A &\mapsto \mathbb{N} \cup \infty \end{aligned}$$

such that $\alpha_S(a) \leq \beta_S(a)$ for all $a \in A$ holds. [[Engel and Hartmann, 1995](#), page 2]

As usual E represent entities, R relationships, and A associations, respectively. The two functions α_S and β_S model the multiplicities for the given SER scheme S and any arc a in the graph.

A solution to a SER scheme in graph form is called a realizer.

A *realizer* is a bipartite digraph $R = (V^R, A^R)$ with the two mappings

$$\begin{aligned} \varphi: V^R &\mapsto V \\ \psi: A^R &\mapsto A \end{aligned}$$

such that

$$\psi(a^R) = (\varphi(e^R), \varphi(r^R)) \quad \text{for all } a^R = (e^R, r^R) \in A^R \quad (2.1)$$

$$\varphi^{-1}(v) \neq \emptyset \quad \text{for all } v \in V \quad (2.2)$$

$$|\varphi^{-1}(v) \cup N(r^R)| = 1 \quad \text{for all } a = (e, r) \in A \text{ and } r^R \in \varphi^{-1}(r) \quad (2.3)$$

$$\alpha_S(a) \leq |\varphi^{-1}(r) \cup N(e^R)| \leq \beta_S(a) \quad \text{for all } a = (e, r) \in A \text{ and } e^R \in \varphi^{-1}(e) \quad (2.4)$$

hold. [[Engel and Hartmann, 1995](#), page 3]

Note that $N(v)$ denotes the neighbourhood of a vertex v , i.e., all those vertices directly connected to v .

Condition 2.1 requires that there must be an homomorphism between the realizer and the SER scheme. Condition 2.2 states that every class and relation must have some instantiation. This is necessary for strong satisfiability. Condition 2.3 guarantees that no direct cycles occur whereas the last one, 2.4, assures that the multiplicities are within range.

As we see these definitions also adhere to the look-here approach common to the ER literature, see e.g., [Lenzerini and Nobili \[1990\]](#). This means we will have to find similar

conditions for the UML case. Anyhow, the presented equations give a good overview how a well-defined formal definition could look like. Further, they unveil the exact differences between their approach in the ER case and our methodology for the look-across style with additional UML features.

Based on Equations 2.1–2.4 Engel and Hartmann [1995] develop an algorithm similar to the method presented by Lenzerini and Nobili [1990] for generating a solution to their system of inequations. The main difference is that they take the logarithm of the path weights in the graph such that they obtain a so-called potential function. For potential functions several results are known, e.g., that a potential function always exists for a strongly connected digraph. Note that the digraph constructed from a SER scheme is always strongly connected if the SER scheme is connected. If not the method can be applied separately to all components of the SER scheme. Another advantage is that the Ford-Bellman [Bellmann, 1958] algorithm can be applied to their formulation. Thus solutions can be computed with a well known procedure.

Further they show that all realizers form a distributive lattice. This means a minimal element always exists. This is what they finally show: they present a slight modification of their Ford-Bellman style algorithm which computes the minimal element.

3. UML Class Diagrams

*Mine is better!
I have many charts and diagrams!*

This chapter describes some aspects of the Unified Modeling Language. We concentrate on UML class diagrams which are useful for modelling in configuration management. We discuss its semantics and deal with its semantic behaviour in particular cases (e.g., with n -ary associations).

3.1. Unified Modeling Language

The *Unified Modeling Language* [Object Management Group, 2005] (UML 2.0) is a standardised specification and modelling language. It was designed to cover a broad range of application areas, and has become the first choice for specification and modelling in software engineering and configuration management. UML is defined using the meta-object facility [Object Management Group, 2002] model and consists of thirteen diagram types listed in Figure 3.1.

Each diagram type covers a distinct usage pattern for specific modelling demands. Figure 3.1 displays their relation in a hierarchical way, where abstract classes are typeset in italics and class diagrams are highlighted in bold face. For an overview see Fowler [2003] and Hitz and Kappel [2003].

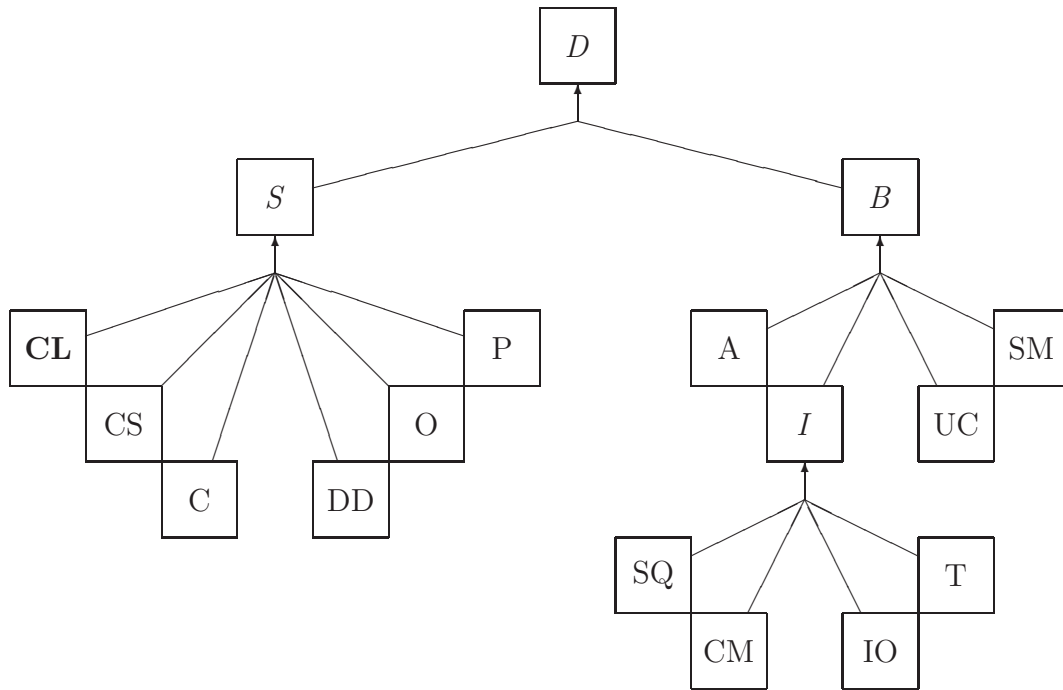
For our purposes—i.e., a specification language for components—we are actually interested in features provided by subsets of UML class diagrams. Class diagrams describe classes, their attributes and methods, and associations between classes in a static context.

Classes: a class is the conceptual representation for an entity to be modelled. It can have attributes and operations. Objects of a given class will be called its instances.

Attributes: each class can have several attributes holding information. Their visibility status can be set to *public*, *protected* and *private*, restricting their access from foreign classes.

Methods: a set of operations may be defined for each class. Typically attribute slots are accessed and modified.

Associations: relations between classes can be modelled by so-called associations. UML defines the concept of *associations*, *aggregations*, *compositions* and *generalisations*. Instances of associations are called links.



A	Activity Diagram	IO	Interaction Overview Diagram
B	<i>Behavior Diagram</i>	O	Object Diagram
CD	Component Diagram	P	Package Diagram
CL	Class Diagram	S	<i>Structure Diagram</i>
CM	Communication Diagram	SM	State Machine Diagram
CS	Composite Structure Diagram	SQ	Sequence Diagram
D	<i>Diagram</i>	T	Timing Diagram
DD	Deployment Diagram	UC	Use Case Diagram
I	<i>Interaction Diagram</i>		

Figure 3.1.: A hierarchy for UML 2.0 diagram types [Object Management Group, 2005, Annex A, page 660]

Many specifications in software engineering and configuration management (e.g., by [Falkner and Fleischanderl \[2001\]](#)) do not use most features available in UML class diagrams. We have found out that class diagrams without attributes and methods restricted to simple associations cover most scenarios in configuration management. This matches those features originally present in ER diagrams for modelling entities and relations.

Thus we will concentrate our work on diagrams consisting of classes with associations between them. A n -ary association links n classes, $n \geq 2$, and has certain properties. At each end is a *multiplicity* and a *uniqueness* constraint. A multiplicity $a..b$ means that the number of partner objects has to be in the interval $[a, b]$. Moreover, each end of an association is marked with the property *unique* (“uniq”, the default) or *non-unique* (“nuniq”). Objects at unique ends are counted only once even if they are connected to a particular object several times. At non-unique ends every connection is counted, even if several of them lead to the same object. Note that the semantics of associations is defined in natural language:

“An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.

When one or more ends of the association have `isUnique=false`, it is possible to have several links associating the same set of instances. In such a case, links carry an additional identifier apart from their end values.

[...]

For an association with N ends, choose any $N-1$ ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection. [...] If the end is marked as unique, this collection is a set; otherwise it allows duplicate elements.” [[Object Management Group, 2005](#), page 37]

Thus it will be necessary to formalise UML’s semantics in a mathematical way in order to prove properties against its definition. This is done in Section 4.1.

Besides the UML infrastructure there exists a formalism for introducing constraints like pre- or post-conditions, or invariants on diagrams: the *Object Constraint Language* [[Object Management Group, 2006](#)] (OCL). Amongst others it is designed for the following purposes [[Object Management Group, 2006](#), page 5]:

- description of pre- or post-conditions for methods,
- specification of class invariants,
- specification of constraints on operations, and
- usage as query language.

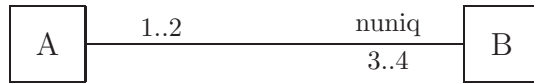


Figure 3.2.: Example of a UML class diagram

We will use OCL only as a supplement for specifying simple constraints on UML class diagrams. This avoids imprecise natural language constraints. In particular we will impose lower bounds on the number of objects that instantiate a given class.

A typical OCL constraint consists of a `context` directive and its constraint corpus (i.e., OCL expression):

```
context ClassName inv: OCLExpression .
```

The context for the OCL expression is set to `ClassName` and one of the stereotypes `inv`, `pre` or `post` defines the constraint type to be an invariant, a precondition or a postcondition, respectively.

OCL expressions typically describe simple Boolean conditions that may contain logical operators and several predefined functions (e.g., `ClassName.allInstances()` returns all instantiated objects of `ClassName`, or `size()` applied to a set returns the number of elements). The keyword `self` refers to an instance of `ClassName`.

As an example consider the diagram in Figure 3.2. It shows a UML diagram typical of configuration specifications. It specifies that every object of class *A* needs to have 3 or 4 links to objects of class *B*, where several *B*-objects may in fact be the same. On the other hand, every object of class *B* must have one or two (different) partners of class *A*.

In addition we might want to specify a lower bound of 4 instantiated objects of class *A*. This can be easily done via an OCL constraint:

```
context A inv: A.allInstances()->size() > 3 .
```

OCL is a very powerful constraint language with a wide variety of specification mechanisms. Most of its features are not used in common modelling specifications. Therefore we will not discuss it in more detail. Refer to the official OCL specification for details.

3.2. Semantics of Multiary Associations

In the last section we saw that UML lacks a clear formal definition for its semantics. The situation gets especially problematic when dealing with unusual modelling scenarios or exotic association types. In this context *n*-ary associations play a special role: on the one hand they are hardly used or only used with trivial multiplicities (i.e., `0..*`), on the other hand they would be very useful in many modelling situations. Such situations could be described far more easily and naturally with *n*-ary associations than with binary ones.

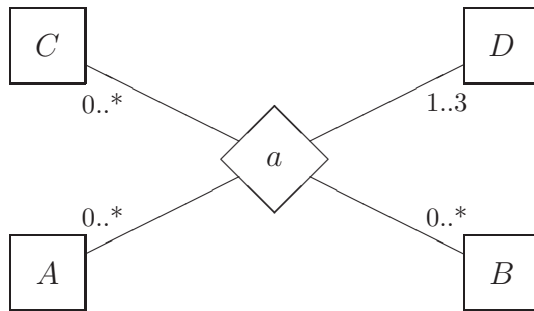


Figure 3.3.: A 4-ary association

Thus n -ary associations could be useful in conceptual modelling with a formal well-defined semantics. There exist two common interpretations in the context of ER and UML associations:

look-across: [Chen \[1976\]](#) used this concept when introducing ER diagrams. Under this interpretation the multiplicity at the end of an association end restrains the number of links for any valid combination of the remaining $n - 1$ partner objects. For [Figure 3.3](#) this means that for any combination of particular objects a , b and c (of classes A , B and C , respectively) the number of links to objects of class D must be between one and three. In other words this interpretation states a global condition for an association by strongly relating all participating classes.

look-here: This interpretation goes back to [Rochfeld \[1986\]](#) and [Rochfeld and Tardieu \[1983\]](#). A multiplicity at one end of an association restricts the number of links to a single instance typed with the class at this association end. For [Figure 3.3](#) this means that for a given object d of class D at least one link and at most three links of association type a must exist. In contrast to the look-across notation this interpretation can be seen as local constraint at each association end.

The natural language definition of multiplicities of UML adheres to the look-across interpretation and has been investigated in detail by several authors. [Génova et al. \[2002\]](#) discuss the meaning of multiplicity of n -ary associations in UML, in particular the semantics of minimum multiplicities in ternary associations, as presented in [Section 3.2.2](#). [Stevens \[2002\]](#) concentrates on the interpretation of binary associations in UML. He focuses on different notions for static and dynamic associations and explores definition ambiguities. He suggests improvements and clarifications to tackle misunderstandings. [Diskin and Dingel \[2006\]](#) present approaches towards formal semantics for associations in UML. They introduce a formal framework, based on sets and mappings, to capture the different interpretations under structural and operational views of UML associations. As we will see on the next few pages, the look-across interpretation introduces

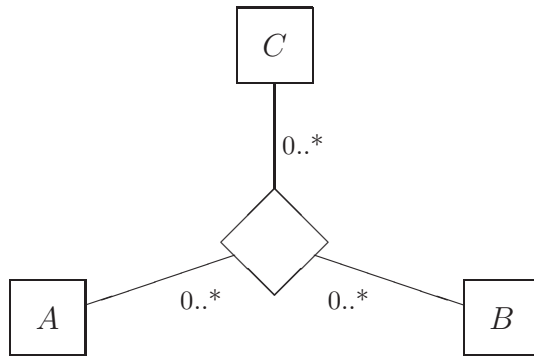


Figure 3.4.: A ternary association interpreted under look-here semantics

several difficulties which prevent the extension of simple mechanisms from binary to n -ary associations.

One of our main considerations will be to deal with consistency and minimality issues in the context of n -ary associations since we believe that n -ary associations can be useful for many purposes. We will define formal semantics based on the informal definition in the UML-standard (see Section 4.1) and present techniques for consistency checks and minimality (see Chapter 6) on this formal groundwork.

3.2.1. Reduction of Multiary to Binary Associations

In the last section we noticed that the look-across and the look-here semantic interpretations imply quite different consequences regarding what multiplicities mean. To avoid problems in the complex n -ary case a promising idea is to transform n -ary associations into binary associations via some reduction.

From the definition of multiplicities in the look-here approach it seems quite obvious how to perform the reduction, since we do not need to care about other participating ends of the association. This means the multiplicities are checked at each end independently of each other, providing a way for modularisation. This idea results in a decomposition, where the n -ary association is replaced by a class which is in turn connected to all original participating entities via binary associations. In other words the new class serves as an intermediary connection mechanism. As an example, Figure 3.4 shows a ternary association. Under the look-here interpretation it can be decomposed into binary associations using a mediator class (Figure 3.5).

Problems arise if we operate under the look-across semantics as used for UML associations. Hartmann [2003] investigates this situation and shows how and why different transformations fail. The above reduction does not work for the look-across interpretation as the multiplicities state a global condition: they require to fix $n - 1$ partner objects and to restrict the number of links to the remaining object(s).

Another possible decomposition is presented in the Figure 3.6. The main disadvantage

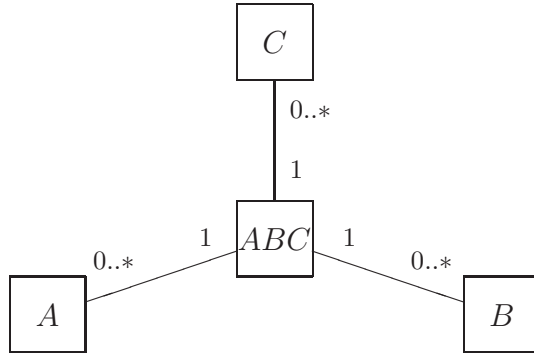


Figure 3.5.: Decomposition of Figure 3.4 into binary associations via mediator class

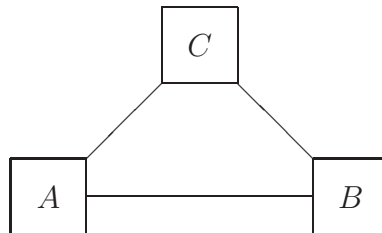


Figure 3.6.: Decomposition into pairwise binary associations

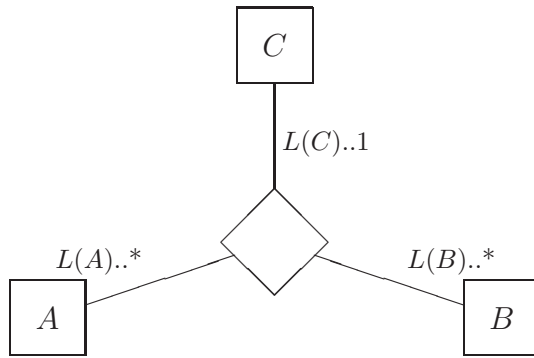


Figure 3.7.: A ternary UML association with generic minimal multiplicities

of this reduction is that the decomposition is not equivalent in general. E.g, consider the tuples

$$\begin{aligned} &(a_1, b_1, c_2) \\ &(a_1, b_2, c_1) \\ &(a_2, b_2, c_1) \\ &(a_2, b_2, c_2) \end{aligned}$$

representing links instantiating the association in Diagram 3.4. Then a decomposition into pairwise binary associations yields

$$\begin{array}{ccc} (a_1, b_1) & (b_1, c_2) & (a_1, c_1) \\ (a_1, b_2) & (b_2, c_1) & (a_1, c_2) \\ (a_2, b_2) & (b_2, c_2) & (a_2, c_1) \\ & & (a_2, c_2) \end{array}$$

The tuple (a_1, b_2, c_2) can be obtained by joining and recomposing the decomposed tuples. But this tuple is not among the original four.

It turns out that a simple reduction from n -ary associations to binary ones under the look-across interpretation is not possible. Only under certain restrictions which were investigated by Jones and Song [2000] and Jones and Song [1996] some reductions are possible. These results rule out a straight-forward generalisation of algorithms for binary UML associations to n -ary ones.

3.2.2. Semantics of Minimum Multiplicity in Ternary UML Associations

Génova et al. [2001] discuss an interesting aspect of n -ary associations which is relevant to a formalisation of UML semantics: the semantics of the minimum multiplicity in ternary associations in UML. At first glance this seems to be a trivial issue since one would expect the UML standard to specify an unambiguous semantics. But it turns out

that three interpretations are compatible with the UML standard. Consider Figure 3.7 which depicts a ternary association with generic (i.e., variable) minimal multiplicities $L(A)$, $L(B)$, and $L(C)$. We compare how these variables are restricted under the different interpretations:

1. Actual tuples [Génova et al., 2001, page 334]:

Under this interpretation only actually existing tuples are considered. In other words a solution consists of several instances of ternary associations of the form

$$\begin{aligned} &(a_1, b_1, c_1) \\ &(a_2, b_2, c_2) \\ &\vdots \end{aligned}$$

Note that under this interpretation an instance of a ternary association must have three components (of the three participating classes). As a consequence,

$$L(A) \neq 0 \wedge L(B) \neq 0 \wedge L(C) \neq 0$$

must hold, also called *zero-forbidden effect*. This means that this interpretation is unsuitable for practical purposes since most examples in the literature with ternary association use 0..* multiplicities.

2. Potential tuples [Génova et al., 2001, page 335]:

The zero-forbidden effect vanishes if the multiplicities are required to hold for all possible tuples, i.e., potential tuples. Non-existing tuples represent cases where one or more components are zero.

A possible problem might be that all potential tuples must get instantiated if one of the minimum multiplicities equals one. E.g., let $L(C) = 1$: this means that any potential pair of instances (a, b) must be connected with exactly one c . Hence we obtain an instantiation of all potential tuples:

$$\begin{aligned} &(a_1, b_2, c_1) \\ &(a_2, b_1, c_1) \\ &\vdots \\ &(a_m, b_n, c_1) \end{aligned}$$

Nevertheless we think this interpretation is preferable since it coincides with the informal (i.e., in natural language described) definition given by the UML specification in Section 3.1.

3. Limping links [Génova et al., 2001, page 335]:

A further interpretation considers the case where an instance of a ternary association is not forced to consist of three elements. Instead we could think of allowing

instances with just two classes and one limping link. Thus we would have links of the form

$$\begin{aligned} &(a_1, b_1, _) \\ &(a_2, _, c_2) \\ &\vdots \end{aligned}$$

The main disadvantage is that it is unclear and undefined what a missing element exactly means. Further it remains ambiguous how many limping links are allowed, e.g., whether

$$(_, b_1, _)$$

is a valid tuple or not.

Therefore this approach keeps to be an academic approach until either practitioners come up with convincing examples or a well-defined semantics is established.

4. A Formal Semantics for UML Class Diagrams

The nice thing about standards is that there are so many of them to choose from.
Andrew S. Tanenbaum

4.1. Configurations and Specifications

To prove the correctness and completeness of our transformation from class diagrams to inequations we need a precise definition of the meaning of UML specifications as well as of configurations satisfying them. Let \mathcal{I} denote the set of intervals over the non-negative integers, i.e.,

$$\mathcal{I} = \{[a, b] \mid a, b \in \mathbb{N}, a \leq b\} \cup \{[a, \infty] \mid a \in \mathbb{N}\} \cup \{[]\},$$

where

$$\begin{aligned} [a, b] &= \{i \mid a \leq i \leq b\} \\ [a, \infty] &= \{i \mid i \geq a\} \end{aligned}$$

and $[]$ is synonymous for the empty set (\emptyset). Moreover, let Cl be a finite set of classes and R be a finite set of roles.

Definition 1 A *specification* is a tuple $\langle A, mult, uniq, lb \rangle$, where:

- $A \subseteq 2^{R \times Cl}$ is the set of associations with the restriction that each role occurs at most once in A and that each association contains at least two elements. Thus, an association $a \in A$ is a finite set of role-class pairs. It is called n -ary if $|a| = n$. Role-class pairs (r, c) are written as $r : c$.
- The function $mult: R \mapsto \mathcal{I}$ associates an interval, called multiplicity, with every role.
- The function $uniq: R \mapsto \{\text{uniq}, \text{nuniq}\}$ fixes the attribute unique/non-unique for every role.
- The function $lb: Cl \mapsto \mathbb{N}$ assigns a non-negative integer to each class giving the lower bound for the number of instantiations.

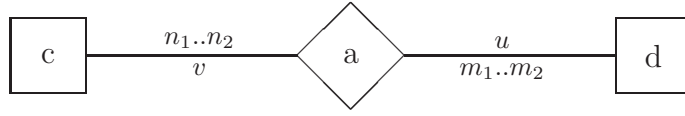
The specification is called *symmetric* if for every association $\{r_1 : c_1, \dots, r_n : c_n\}$ in A we have $uniq(r_1) = \dots = uniq(r_n)$.

An association can be viewed as a hyper-edge connecting several classes via arcs labelled with roles. Alternatively, $(Cl \cup A, R)$ can be viewed as a bi-partite graph, where roles connect reified associations with classes.

Given an association $a = \{r_1 : c, r_2 : d\}$, the situation

$$\begin{array}{lcl} mult(r_1) = [n_1, n_2] & \text{and} & uniq(r_1) = v \\ mult(r_2) = [m_1, m_2] & & uniq(r_2) = u \end{array}$$

is depicted as



Binary associations are usually depicted as



that means the association is implicit in the graphical representation of a binary association.

Definition 2 A *configuration* is a tuple $\langle O, L, class, ass, obj \rangle$, where:

- O is a set of objects.
- L is a set of links.
- The function $class: O \mapsto Cl$ maps each object to its class; we say that object o is of class c if $class(o) = c$.
- The function $ass: L \mapsto A$ maps each link to its association.
- The function $obj: L \mapsto 2^{R \times O}$ maps each link to the objects it connects via particular roles such that

$$|ass(l)| = |obj(l)| \quad \text{and} \quad ass(l) = \{r_1 : class(o_1), \dots, r_n : class(o_n)\}$$

holds for all links $l \in L$, where $obj(l) = \{r_1 : o_1, \dots, r_n : o_n\}$.

Note that an association is uniquely identified by the set $\{r_1 : c_1, \dots, r_n : c_n\}$ (or, in fact, by any single role occurring in it), whereas we may have different links l_1, l_2 instantiating the same association and connecting the same objects, i.e., we may have $obj(l_1) = obj(l_2)$ but $l_1 \neq l_2$.

Let $C = \langle O, L, class, ass, obj \rangle$ be a configuration. The *cardinality* $|c|_C$ of a class c is $|class^{-1}(c)|$, i.e., the number of objects in the configuration that are of class c . If C is clear from context we omit the subscript. A configuration C is smaller than or equal to a configuration D , written as $C \leq D$, if $|c|_C \leq |c|_D$ for all $c \in Cl$.

Let x denote a partial link, i.e., $x = \{r_1 : o_1, \dots, r_k : o_k\}$, typically with k being $n - 1$. We use $\delta(x)$ to denote the number of links connecting the objects in x , and $\gamma_r(x)$ to denote the number of different objects occupying role r in the links containing x . Formally:

$$\begin{aligned}\delta(x) &= |\{l \in L \mid x \subseteq obj(l)\}|, \\ \gamma_r(x) &= |\{o \in O \mid x \cup \{r : o\} \subseteq obj(l) \text{ for some } l \in L\}|.\end{aligned}$$

Moreover, let a^O denote the set of potential links connecting objects in O that are well-typed with respect to a single association $a = \{r_1 : c_1, \dots, r_n : c_n\}$, and let A^O denote the potential links well-typed with respect to some association in A :

$$\begin{aligned}\{r_1 : c_1, \dots, r_n : c_n\}^O &= \{\{r_1 : o_1, \dots, r_n : o_n\} \mid o_i \in O, class(o_i) = c_i \text{ for } i = 1, \dots, n\} \\ A^O &= \bigcup_{a \in A} a^O\end{aligned}$$

Definition 3 A configuration $\langle O, L, class, ass, obj \rangle$ is said to *satisfy* a specification $\langle A, mult, uniq, lb \rangle$ if for all classes $c \in Cl$

- $|c| \geq lb(c)$

holds and for all roles $r \in R$ and all potential links $p \in A^O$ such that $r : o$ occurs in p (for some object o) the following conditions hold:

- $\delta(p \setminus \{r : o\}) \in mult(r)$ if $uniq(r) = \text{nuniq}$, and
- $\gamma_r(p \setminus \{r : o\}) \in mult(r)$ if $uniq(r) = \text{uniq}$.

A specification is *weakly consistent* if it is satisfied by some configuration, and is *strongly consistent* if for each class c there exists a configuration such that $|c| > 0$. If a specification is not weakly consistent it is called *inconsistent*.

Note that strong consistency implies weak consistency. If all lower bounds are zero then a specification is trivially weakly consistent, since it is satisfied by the configuration containing no objects.

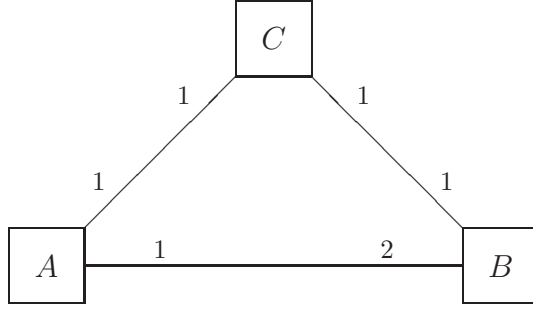


Figure 4.1.: UML specification that is weakly but not strongly consistent

Example 4 Figure 4.1 shows a symmetric specification that is weakly but not strongly consistent: Every object of class A requires two objects of class B , each of which is uniquely associated with an object of class C , which in turn correspond to exactly one object of class A . But each of these two A s requires two objects of class B , and so on. Hence the only configuration satisfying the specification is the trivial one. If we additionally impose the constraint that there has to be at least one object, it becomes inconsistent.

Example 5 Let S be the asymmetric specification depicted in Figure 3.2 with the assumption $lb(A) = lb(B) = 0$. According to the mapping between the graphical representation of UML diagrams and their formal description on page 30 we obtain an association $a = \{r_1 : A, r_2 : B\}$ with

$$\begin{aligned} mult(r_1) &= [1, 2] \\ mult(r_2) &= [3, 4] \end{aligned}$$

and

$$\begin{aligned} uniq(r_1) &= \text{uniq} \\ uniq(r_2) &= \text{muniq} . \end{aligned}$$

Furthermore, let $C = \langle O, L, class, ass, obj \rangle$ be the configuration defined by

$$\begin{aligned} O &= \{o_1, o_2\} \\ L &= \{l_1, l_2, l_3\} \\ class(o_1) &= A \\ class(o_2) &= B \\ ass(l_1) &= ass(l_2) = ass(l_3) = a \\ obj(l_1) &= obj(l_2) = obj(l_3) = \{r_1 : o_1, r_2 : o_2\} . \end{aligned}$$

C satisfies S , since we have

$$\begin{aligned}\gamma_{r_1}(\{r_2 : o_2\}) &= |\{o_1\}| = 1 \in [1, 2] = \text{mult}(r_1) \quad \text{and} \\ \delta(\{r_1 : o_1\}) &= |\{l_1, l_2, l_3\}| = 3 \in [3, 4] = \text{mult}(r_2) \quad .\end{aligned}$$

If all ends of an association have the attribute `uniq`, then multiple links (links with the same `obj`-value) between the corresponding objects can be treated as a single one since they cannot be distinguished. One link is as good as many: the multiplicities restrict only the number of different objects. Therefore we concentrate on *normalised* configurations which contain no multiple links for associations where all roles are tagged as unique.

For symmetric specifications and normalised configurations the satisfiability conditions can be simplified.

Proposition 6 *A normalised configuration satisfies a symmetric specification if and only if for all classes $c \in Cl$*

- $|c| \geq lb(c)$

holds and for all roles $r \in R$ and all potential links $p \in A^O$ such that $r : o$ occurs in p (for some object o) the following conditions hold:

- $\delta(p \setminus \{r : o\}) \in \text{mult}(r)$ and
- if $\text{uniq}(r) = \text{uniq}$ then for all role-object tuples x, y and all links l_1, l_2 such that $\text{obj}(l_1) = \{r : o, x\}$ and $\text{obj}(l_2) = \{r : o, y\}$, $l_1 \neq l_2$ implies $x \neq y$.

By this proposition we may treat all associations as if being labelled `nuniq`; we have only to make sure that any two objects instantiating `uniq-uniq` associated classes are connected by at most one link.

Example 7 This uniform view of `uniq/nuniq` does not work for asymmetric specifications. Consider the specification and configuration of Example 5, where $\text{uniq}(r_1) = \text{uniq}$ but $\text{uniq}(r_2) = \text{nuniq}$. Let

$$\begin{aligned}r : o &= r_2 : o_2 \\ x = y &= r_1 : o_1 \quad .\end{aligned}$$

Then we have

$$\begin{aligned}\text{obj}(l_1) &= \{r : o, x\} \\ \text{obj}(l_2) &= \{r : o, y\}\end{aligned}$$

and $l_1 \neq l_2$ but $x = y$. Note that we need all three links; removing any of the multiple links would lead to a configuration not satisfying the specification anymore.

5. UML Class Specifications with Binary Associations

*White dwarf seeks red giant for binary relationship.
sci.astro newsgroup*

In this chapter we will show a translation from UML specifications to linear Diophantine inequations. We will present an approach based on weighted directed graphs to solve these inequations efficiently and show a way to compute the minimal solutions.

5.1. From Specifications to Linear Diophantine Inequations

In this section we translate specifications containing binary associations (symmetric or mixed) to certain inequations and show that a specification is consistent if and only if the corresponding inequations are solvable. Moreover, the solutions of the inequations describe all satisfying configurations.

For each class $c \in Cl$, let x_c be a variable ranging over the non-negative integers. Let $\{r_1 : c, r_2 : d\}$ be an association with $mult(r_2) = [m_1, m_2]$ and $mult(r_1) = [n_1, n_2]$. We define the following abbreviations:

$$\begin{aligned}\varphi_{lb}(c) &:= x_c \geq lb(c) \\ \varphi_{mult}(r_1:c, r_2:d) &:= m_2 \cdot x_c \geq n_1 \cdot x_d \\ \varphi_{uniq_min}(r_1:c, r_2:d) &:= (x_d > 0 \implies x_c \geq n_1) \\ \varphi_{uniq_max}(r_1:c, r_2:d) &:= (m_1 > 0 \implies n_2 \cdot x_d \geq x_c)\end{aligned}$$

where \implies denotes logical implication. Next we define formulas corresponding to the three types of binary associations, namely nuniq/nuniq, uniq/uniq and uniq/nuniq:

$$\begin{aligned}\psi_{nn}(x, y) &:= \varphi_{mult}(x, y) \wedge \varphi_{mult}(y, x) \\ \psi_{uu}(x, y) &:= \varphi_{mult}(x, y) \wedge \varphi_{uniq_min}(x, y) \wedge \\ &\quad \varphi_{mult}(y, x) \wedge \varphi_{uniq_min}(y, x) \\ \psi_{un}(x, y) &:= \varphi_{mult}(x, y) \wedge \varphi_{uniq_min}(x, y) \wedge \varphi_{uniq_max}(x, y)\end{aligned}$$

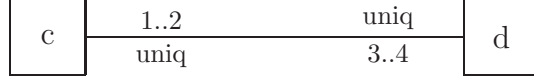


Figure 5.1.: UML specification for Example 9

Definition 8 The *satisfiability condition (sat-condition)* for a specification S , denoted by $sc(S)$, is the formula

$$\bigwedge_{c \in Cl} \varphi_{lb}(c) \wedge \bigwedge_{\substack{\{r_1:c, r_2:d\} \in A \\ uniq(r_1)=nuniq \\ uniq(r_2)=nuniq}} \psi_{nn}(r_1:c, r_2:d) \wedge \bigwedge_{\substack{\{r_1:c, r_2:d\} \in A \\ uniq(r_1)=uniq \\ uniq(r_2)=uniq}} \psi_{uu}(r_1:c, r_2:d) \wedge \bigwedge_{\substack{\{r_1:c, r_2:d\} \in A \\ uniq(r_1)=uniq \\ uniq(r_2)=nuniq}} \psi_{un}(r_1:c, r_2:d)$$

An *interpretation* for the formula $sc(S)$ is a mapping assigning a non-negative integer to each variable in $sc(S)$. A *solution* of $sc(S)$ is an interpretation satisfying the formula $sc(S)$.

For interpretations σ we will also write $\sigma(c)$ instead of $\sigma(x_c)$ since there is a one-to-one mapping between classes and variables. Note that the formulas corresponding to symmetric associations occur twice in $sc(S)$, since we have

$$\begin{aligned} \psi_{nn}(r_1:c, r_2:d) &= \psi_{nn}(r_2:d, r_1:c) \\ \psi_{uu}(r_1:c, r_2:d) &= \psi_{uu}(r_2:d, r_1:c) . \end{aligned}$$

Example 9 Let S be the specification $\langle A, mult, uniq, lb \rangle$ where

$$\begin{aligned} A &= \{\{r_1 : c, r_2 : d\}\} \\ mult(r_1) &= [1, 2] \\ mult(r_2) &= [3, 4] \\ uniq(r_1) &= uniq \\ uniq(r_2) &= uniq \\ lb(c) &= 1 \\ lb(d) &= 0 . \end{aligned}$$

Figure 5.1 depicts this situation.

Then the sat-condition $sc(S)$ is the conjunction of

$$\begin{array}{llll} x_c \geq 1 & 2 \cdot x_d \geq 3 \cdot x_c & x_c > 0 & \implies x_d \geq 3 \\ x_d \geq 0 & 4 \cdot x_c \geq 1 \cdot x_d & x_d > 0 & \implies x_c \geq 1 . \end{array}$$

Remark 10 For each pair of classes c and d of a symmetric association, the sat-condition contains the constraints φ_{mult} , which express that the intervals $[m_1, m_2] \cdot x_c$

and $[n_1, n_2] \cdot x_d$ overlap, i.e., they are satisfied if and only if there is a number k such that

$$\begin{aligned} m_1 \cdot x_c &\leq k \leq m_2 \cdot x_c \\ n_1 \cdot x_d &\leq k \leq n_2 \cdot x_d \end{aligned}$$

hold.

Example 11 Consider the specification in Example 5. The sat-condition $sc(S)$ is the conjunction of

$$\begin{aligned} x_c &\geq 0 & 2 \cdot x_d &\geq x_c & x_d > 0 &\implies x_c \geq 1 . \\ x_d &\geq 0 & 4 \cdot x_c &\geq 1 \cdot x_d & & \end{aligned}$$

Remark 12 For each pair of classes c and d of an asymmetric association (w.l.o.g. assume $uniq(r_1) = \text{uniq}$), the sat-condition contains the constraints

$$\begin{aligned} m_2 \cdot x_c &\geq n_1 \cdot x_d \\ n_2 \cdot x_d &\geq x_c , \end{aligned}$$

which express that the intervals $[1, m_2] \cdot x_c$ and $[n_1, n_2] \cdot x_d$ overlap, i.e., they are satisfied if and only if there is a number k such that

$$\begin{aligned} x_c &\leq k \leq m_2 \cdot x_c \\ n_1 \cdot x_d &\leq k \leq n_2 \cdot x_d \end{aligned}$$

hold.

Theorem 13 *A specification S is weakly consistent if and only if the formula $sc(S)$ is solvable.*

Proof We prove the two directions separately.

(\implies) Let the specification $S = \langle A, mult, uniq, lb \rangle$ be weakly consistent, and let $C = \langle O, L, class, ass, obj \rangle$ be a configuration satisfying S . W.l.o.g. we assume that C is normalised, i.e., that it does not contain multiple links corresponding to uniq-uniq associations. We show that the mapping σ defined by $\sigma(x_c) = |c|$ for all classes c satisfies the specification S .

The lower bound constraints $x_c \geq lb(c)$ hold since we have $|c| \geq lb(c)$ for all classes c .

For the symmetric case (i.e., $uniq(r_1) = \text{uniq}(r_2)$) now consider two classes c and d connected by an association $\{r_1 : c, r_2 : d\}$. Let k be the number of all links connecting objects of class c with objects of class d . Because of

$$\begin{aligned} \delta(\{r_1 : o\}) &\in mult(r_2) = [m_1, m_2] \\ \delta(\{r_2 : p\}) &\in mult(r_1) = [n_1, n_2] \end{aligned}$$

for all objects o of type c and objects p of type d , we have

$$\begin{aligned} m_1 \cdot x_c &\leq k \leq m_2 \cdot x_c \\ n_1 \cdot x_d &\leq k \leq n_2 \cdot x_d . \end{aligned}$$

By Remark 10, the existence of such a number k is expressed by the constraints φ_{mult} . Finally, if $\text{uniq}(r_2) = \text{uniq}$ (similar argumentation for $\text{uniq}(r_1)$) holds then any object o of class c has to be connected to at least m_1 different objects of class d , i.e., if the number of c -objects is greater than zero, then the number of d -objects has to be greater than m_1 :

$$x_c > 0 \implies x_d \geq m_1 .$$

For the asymmetric case (w.l.o.g. assume $\text{uniq}(r_1) = \text{uniq}$) let k denote the number of allowed links between all objects of class c and all objects of class d . Because of

$$\begin{aligned} \delta(\{r_1 : o\}) &\in \text{mult}(r_2) = [m_1, m_2] \\ \gamma_{r_1}(\{r_2 : p\}) &\in \text{mult}(r_1) = [n_1, n_2] \end{aligned}$$

for all objects o of type c and objects p of type d , following conditions hold:

$$\begin{aligned} m_2 \cdot x_c &\geq n_1 \cdot x_d \\ n_2 \cdot x_d &\geq x_c \\ x_d > 0 &\implies x_c \geq n_1 . \end{aligned}$$

Note that it is essential for mixed associations to handle multiple links also for uniq ends (i.e., there is no normalisation between mixed associations), as explained in Example 7. The first line models the constraint for the nuniq end, similar to one of the φ_{mult} equations in the symmetric case. The second line enforces that there are not more objects of class c than the maximal amount of links from objects of class d to objects of class c permits. The third equation models the minimum for the number of different objects of class c , similar to the symmetric case.

(\Leftarrow) Given a specification S and a solution σ of the sat-condition $\text{sc}(S)$ we construct a configuration $C = \langle O, L, \text{class}, \text{ass}, \text{obj} \rangle$ that satisfies S .

For all classes $c \in Cl$, we define O and class such that O contains exactly $\sigma(c)$ objects of class c . We construct L and obj according to Algorithm 1. By Remark 10 and Remark 12, the number k chosen in lines 5–11 exists since the intersection is non-empty. The while-loop distributes the k links sequentially in a uniform manner, i.e., before inserting the link we have

$$\begin{aligned} \delta(\{r_c : o\}) &= \lfloor k' / \sigma(c) \rfloor \\ \delta(\{r_d : p\}) &= \lfloor k' / \sigma(d) \rfloor . \end{aligned}$$

Note that if line 16 for the unique-unique case gets triggered we have

$$\delta(\{r_c : o\}) = \delta(\{r_c : o'\}) = k' / \sigma(c)$$

for all objects o, o' of class c . As the links are filled up uniformly between objects, this guarantees that no link between $o_i, p_{(j+1) \bmod b}$ already exists. Thus for the unique-unique case we obtain that for all objects o_1, o_2 of class c and all links l_1, l_2 such that

$$\begin{aligned} \text{obj}(l_1) &= \{r_c : o, r_d : o_1\} \\ \text{obj}(l_2) &= \{r_c : o, r_d : o_2\} \end{aligned}$$

$l_1 \neq l_2$ implies $r_d : o_1 \neq r_d : o_2$.

Therefore at the end of the loop the condition

$$\lfloor k/\sigma(c) \rfloor \leq \delta(r_c : o) \leq \lceil k/\sigma(c) \rceil$$

holds. By the choice of k in the symmetric case we have

$$\sigma(c) \cdot m_1 \leq k \leq \sigma(c) \cdot m_2 \quad ,$$

i.e.,

$$m_1 \leq \lfloor k/\sigma(c) \rfloor \leq k/\sigma(c) \leq \lceil k/\sigma(c) \rceil \leq m_2 \quad .$$

Combining the two chains of inequations we obtain

$$m_1 \leq \delta(\{r_c : o\}) \leq m_2 \quad ,$$

i.e.,

$$\delta(\{r_c : o\}) \in \text{mult}(r_d) \quad .$$

By a dual argument we derive

$$\delta(\{r_d : p\}) \in \text{mult}(r_c)$$

and its uniq constraints.

For the asymmetric case line 16 never gets triggered. Thus the links are filled up completely uniformly and we have

$$\lfloor k/\sigma(d) \rfloor \leq \delta(r_d : p) \leq \lceil k/\sigma(d) \rceil \quad .$$

By the choice of k the inequation $n_1 \leq k/\sigma(d)$ holds. Combining both inequations we obtain

$$n_1 \leq \delta(\{r_d : p\}) \quad .$$

As $x_d > 0 \implies x_c \geq n_1$ holds since σ must also satisfy this constraint, we know that there are at least n_1 distinct objects of class c . By the strictly uniform distribution of links we obtain

$$n_1 \leq \gamma_{r_c}(\{r_d : p\}) \quad ,$$

i.e., the lower bound of $\text{mult}(r_c)$. For proving the upper bound we note that each of the $\sigma(d)$ objects of class d is connected (due to uniformity) with

$$\lfloor k/\sigma(c) \rfloor \leq l_c \leq \lceil k/\sigma(c) \rceil$$

links to objects of class c . Further there are

$$\lfloor k/\sigma(d) \rfloor \leq l_d \leq \lceil k/\sigma(d) \rceil$$

links outgoing from object p . Thus the number of different objects connected to p by role r_d (which in fact is $\delta(\{r_d : p\})$) is l_d/l_c , i.e.,

$$\frac{k/\sigma(d)}{k/\sigma(c)} = \frac{\sigma(c)}{\sigma(d)} .$$

By constraint $\varphi_{\text{uniq_max}}$, i.e., $n_2 \cdot x_d \geq x_c$, this ratio is smaller than n_2 , which is the upper bound. So we have

$$\gamma_{r_c}(\{r_d : p\}) \in \text{mult}(r_c) = [n_1, n_2] .$$

For the non-unique end, the condition

$$\delta(\{r_c : o\}) \in \text{mult}(r_d)$$

is derived as in the symmetric case. By dual arguments we obtain

$$\gamma_{r_d}(\{r_c : o\}) \in \text{mult}(r_d)$$

and

$$\delta(\{r_d : p\}) \in \text{mult}(r_c)$$

for the dual non-unique–unique case.

We conclude that the configuration C satisfies the specification S . □

5.2. Solving Linear Diophantine Inequations

Several algorithms have been proposed for solving linear inequations over non-negative integers. Based on the results on the solution of linear Diophantine equations by [Clausen and Fortenbacher \[1989\]](#) and [Contejean and Devie \[1994\]](#), an algorithm without slack variables is presented by [Ajili and Contejean \[1997\]](#). With small modifications this algorithm can be used to solve the inequations obtained in the last section. Unfortunately, it has exponential run time in general. This complexity is partly intrinsic to the problem: [Lagarias \[1985\]](#) shows that the problem 2-IP (integer programming with two variables) is NP-complete. More precisely, 2-IP is defined as follows: Given a finite set of integer triples

$$\{(a_{1,k}, a_{2,k}, a_{3,k}) \mid 1 \leq k \leq M\}$$

and two functions

$$\begin{aligned} \sigma &: \{1, 2, \dots, M\} \rightarrow \{1, 2, \dots, N\} \\ \tau &: \{1, 2, \dots, M\} \rightarrow \{1, 2, \dots, N\} \end{aligned}$$

Algorithm 1 Constructing the links of a configuration

```
1: let  $L = \emptyset$ 
2: for all sets  $\{c, d\} \in 2^{Cl}$  with  $\{r_c : c, r_d : d\} \in A$  do
3:   let  $[m_1, m_2] = \text{mult}(r_d)$  and  $[n_1, n_2] = \text{mult}(r_c)$ 
4:   let  $o_0, \dots, o_{a-1}$  be the objects of class  $c$ 
5:   let  $p_0, \dots, p_{b-1}$  be the objects of class  $d$ 
6:   if  $\text{uniq}(r_c) = \text{uniq} \neq \text{uniq}(r_d)$  then
7:     choose a number  $k \in [\sigma(c), \sigma(c) \cdot m_2] \cap [\sigma(d) \cdot n_1, \sigma(d) \cdot n_2]$ 
8:   else if  $\text{uniq}(r_c) = \text{nuniq} \neq \text{uniq}(r_d)$  then
9:     choose a number  $k \in [\sigma(c) \cdot m_1, \sigma(c) \cdot m_2] \cap [\sigma(d), \sigma(d) \cdot n_2]$ 
10:  else
11:    choose a number  $k \in [\sigma(c) \cdot m_1, \sigma(c) \cdot m_2] \cap [\sigma(d) \cdot n_1, \sigma(d) \cdot n_2]$ 
12:  end if
13:   $k' \leftarrow 0$ 
14:   $i \leftarrow 0$ 
15:   $j \leftarrow 0$ 
16:  while  $k' < k$  do
17:    if  $\text{uniq}(r_c) = \text{uniq}(r_d) = \text{uniq}$ 
18:      and there is a link  $l \in L$  such that  $\text{obj}(l) = \{r_c : o_i, r_d : p_j\}$  then
19:         $j \leftarrow (j + 1) \bmod b$ 
20:      end if
21:      add a link  $l$  with  $\text{obj}(l) = \{r_c : o_i, r_d : p_j\}$  to  $L$ 
22:       $k' \leftarrow k' + 1$ 
23:       $i \leftarrow (i + 1) \bmod a$ 
24:       $j \leftarrow (j + 1) \bmod b$ 
25:    end while
26:  end for
27: return  $L$ 
```

the problem consists in determining whether the integer program

$$a_{1,k} \cdot x_{\sigma(k)} + a_{2,k} \cdot x_{\tau(k)} \leq a_{3,k}$$

for $1 \leq k \leq M$ has an integer solution $\{x_i \mid 1 \leq i \leq N\}$ satisfying all inequations.

Thus the algorithm presented by [Ajili and Contejean \[1997\]](#) cannot be polynomial in the worst case. In fact, the algorithm has the high pruning bound of

$$m_B(q + m_B - r) \left(\frac{\sum_{i,j} |a_{ij}| + \sum_{i,j} |b_{ij}| + m_B}{r} \right)^{2r},$$

where m_B denotes the number of lines for the matrix B that describes the input inequations, q is the number of unknowns in the system of m constraints, r is the rank of matrix $\begin{pmatrix} A & 0 \\ B & I \end{pmatrix}$, where A describes the input equations, and a_{ij} and b_{ij} are the entries of A and B , respectively.

However, our class of inequations has a specific property: we need no upper bounds, i.e., no inequation is of the form $c \geq ax + by$ or $c \geq ax$. We show in the following that in this case solvability can be checked in polynomial time. Moreover, the solutions are closed under the minimum operation and under linear combinations; hence the minimal solution is unique.

Definition 14 A UML-constraint is either

- an inequation of the form $b \cdot y \geq a \cdot x$,
- an inequation $y \geq a$,
- an implication of the form $a > 0 \implies b \cdot y \geq x$ or
- an implication of the form $x > 0 \implies y \geq a$,

where a and b are positive integers and x and y are variables. A UML-formula is a conjunction of UML-constraints.

Note that an inequation

$$a_1 > 0 \implies b \cdot y \geq x$$

can be transformed to

$$b \cdot y \geq a_2 \cdot x$$

(in this special case with $a_2 = 1$) by testing whether the constant a_1 is positive. This simplification can be done in a preprocessing step since the premise of the implication contains no variable.

Example 15 Consider the sat-condition of [Example 11](#). Each inequation in it is a UML-constraint, since

$$\begin{aligned} x_c &\geq 0 \\ x_d &\geq 0 \end{aligned}$$

are of the form $y \geq a$,

$$2 \cdot x_d \geq x_c$$

is of the form $a > 0 \implies b \cdot y \geq x$ (after simplification),

$$4 \cdot x_c \geq 1 \cdot x_d$$

is of the form $b \cdot y \geq a \cdot x$, and finally

$$x_d > 0 \implies x_c \geq 1$$

is of the form $x > 0 \implies y \geq a$.

Let σ and τ be interpretations for a UML-formula φ , i.e., σ and τ map the variables in φ to non-negative integers, and let a be a non-negative integer. We define the minimum, sum, and scalar product of interpretations for all variables x occurring in φ as follows:

$$\begin{aligned} \min(\sigma, \tau)(x) &= \min(\sigma(x), \tau(x)) \\ (\sigma + \tau)(x) &= \sigma(x) + \tau(x) \\ (a\sigma)(x) &= a\sigma(x) \end{aligned}$$

Moreover, we define $\sigma \leq \tau$ as $\sigma(x) \leq \tau(x)$ for all variables x .

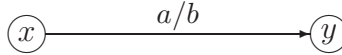
Proposition 16 *Let φ be a UML-formula, and let σ and τ be solutions of φ . Then $\min(\sigma, \tau)$, $\sigma + \tau$ and $a\sigma$ are also solutions of φ .*

Corollary 17 *The minimal solution of a UML-formula is unique.*

Corollary 18 *If a UML-formula has a non-trivial solution, then it has infinitely many solutions.*

We now present our algorithm for checking the consistency of UML-formulas. We call a variable x *inconsistent* if $\sigma(x) = 0$ holds for all solutions σ of a formula. The class corresponding to an inconsistent variable will be empty in all satisfying configurations.

Our algorithm has to pinpoint all inconsistent variables and should provide an explanation for the inconsistencies. The core algorithm operates on a weighted directed graph constructed from the UML-formula φ . The variables in φ form the nodes of the graph. There is an edge from node x to node y with weight a/b iff φ contains an inequation $b \cdot y \geq a \cdot x$:



Theorem 19 *Let I be a set of inequations of the form $b \cdot y \geq a \cdot x$, and let V be the set of variables occurring in I . Algorithm 2 determines the path with the maximal weight for each pair of nodes in the graph representation of I in time $O(|I| \cdot |V|^2)$.*

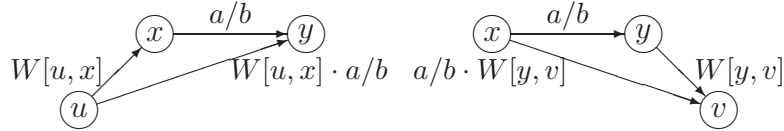
Algorithm 2 Compute paths with maximal weights

```
1: Let  $I$  be a set of input inequations  $b \cdot y \geq a \cdot x$ 
2: Let  $V$  be the set of variables occurring in  $I$ 
3: Let  $W$  be a  $|V| \times |V|$  matrix of non-negative rational numbers, initially set to zero.
4: Let  $P$  be a  $|V| \times |V|$  matrix of strings of variables, initially set to the empty strings.

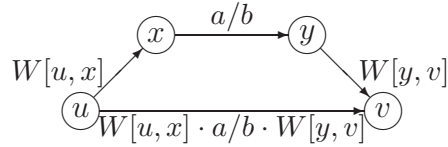
5: for all inequations  $b \cdot y \geq a \cdot x$  in  $I$  do
6:    $w \leftarrow a/b$ 
7:   if  $w > W[x, y]$  then
8:      $W[x, y] \leftarrow w$ 
9:      $P[x, y] \leftarrow y$ 
10:  end if
11:  for all variables  $u$  in  $V$  such that  $u \neq x$  do
12:     $w \leftarrow W[u, x] \cdot a/b$ 
13:    if  $w > W[u, y]$  and  $y \notin P[u, x]$  then
14:       $W[u, y] \leftarrow w$ 
15:       $P[u, y] \leftarrow P[u, x].y$ 
16:    end if
17:  end for
18:  for all variables  $v$  in  $V$  such that  $v \neq y$  do
19:     $w \leftarrow a/b \cdot W[y, v]$ 
20:    if  $w > W[x, v]$  and  $y \notin P[y, v]$  then
21:       $W[x, v] \leftarrow w$ 
22:       $P[x, v] \leftarrow y.P[y, v]$ 
23:    end if
24:  end for
25:  for all variables  $u, v$  in  $V$  such that  $u \neq x$  and  $v \neq y$  do
26:     $w \leftarrow W[u, x] \cdot a/b \cdot W[y, v]$ 
27:    if  $w > W[u, v]$  and the strings  $P[u, x]$ ,  $y$ , and  $P[y, v]$  are pairwise disjoint then
28:       $W[u, v] \leftarrow w$ 
29:       $P[u, v] \leftarrow P[u, x].y.P[y, v]$ 
30:    end if
31:  end for
32: end for
33: return  $(W, P)$ 
```

Proof Algorithm 2 processes the inequations incrementally and saves the maximal weight and path between variables at each step. Lines 6–10 update the weight matrix W and path matrix P concerning the two directly involved variables.

In lines 11–17 all variables u with a path to y are checked whether there is a heavier path from u to y via x . Note that we need not check variables more often, as we save the maximal weights in the weight matrix W . Further we avoid intermediate cycles by requiring $y \notin P[u, x]$. If the new path weight is greater, update both W and P . In a dual manner the lines 18–24 update the weights for paths from x to v via y .



Finally in lines 25–32 all paths from u to v are updated, if there is a new, heavier path using the edge from x to y .



The algorithm obviously terminates because all loops are bounded. Counting the iterations we obtain as time complexity $O(|I| \cdot (1 + |V| + |V| + |V| \cdot |V|))$, i.e. $O(|I| \cdot |V|^2)$. \square

Theorem 20 *Let W be the weight matrix returned by Algorithm 2. A variable x is inconsistent if and only if there is a node y such that $W[x, y] > 0$ and $W[y, y] > 1$ both hold.*

Proof $W[x, y] > 0$ and $W[y, y] > 0$ mean that the inequalities

$$y \geq W[y, y] \cdot y \geq W[x, y] \cdot x$$

are contained in the transitive closure of I . Because of $W[y, y] > 1$ the first one can only be satisfied by setting y to zero. This implies that x has to be zero, too. \square

For a given specification we can check its sat-condition by Theorem 20. Note that the constraints φ_{lb} and $\varphi_{\text{uniq_min}}$ do not affect strong consistency since they only fix lower bounds, which can always be satisfied by a configuration provided the corresponding variable is consistent.

Example 21 Have a look at the specification S depicted in Figure 4.1. It has three associations

$$\begin{aligned} a_{AB} &= \{r_{AB} : A, r_{BA} : B\} \\ a_{BC} &= \{r_{BC} : B, r_{CB} : C\} \\ a_{AC} &= \{r_{AC} : A, r_{CA} : C\} \end{aligned}$$

with the multiplicities

$$\begin{array}{ll} \text{mult}(r_{AB}) = [1, 1] & \text{mult}(r_{CB}) = [1, 1] \\ \text{mult}(r_{BA}) = [2, 2] & \text{mult}(r_{AC}) = [1, 1] \\ \text{mult}(r_{BC}) = [1, 1] & \text{mult}(r_{CA}) = [1, 1] , \end{array}$$

and the uniqueness constraints $\text{uniq}(r) = \text{uniq}$ for all roles r . Further assume that

$$\begin{array}{l} \text{lb}(A) = 1 \\ \text{lb}(B) = 0 \\ \text{lb}(C) = 0 \end{array}$$

holds. We obtain the following sat-condition $\text{sc}(S)$

$$\begin{array}{lll} x_A \geq 1 & 1 \cdot x_B \geq 2 \cdot x_A & x_A > 0 \implies x_B \geq 2 \\ x_B \geq 0 & 2 \cdot x_A \geq 1 \cdot x_B & x_B > 0 \implies x_A \geq 1 \\ x_C \geq 0 & 1 \cdot x_C \geq 1 \cdot x_B & x_B > 0 \implies x_C \geq 1 \\ & 1 \cdot x_B \geq 1 \cdot x_C & x_C > 0 \implies x_B \geq 1 \\ & 1 \cdot x_C \geq 1 \cdot x_A & x_A > 0 \implies x_C \geq 1 \\ & 1 \cdot x_A \geq 1 \cdot x_C & x_C > 0 \implies x_A \geq 1 . \end{array}$$

Let us now check the specification for consistency as described in the proof of Theorem 20 using Algorithm 2. At the beginning we have

$$W = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{array}{c} \text{C} \\ \swarrow \quad \searrow \\ \text{A} \quad \longleftrightarrow \quad \text{B} \\ \swarrow \quad \searrow \\ \text{A} \quad \longleftrightarrow \quad \text{B} \end{array} \quad P = \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

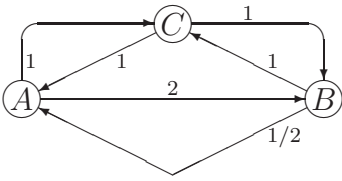
After two iterations for the inequations $1 \cdot x_B \geq 2 \cdot x_A$ and $2 \cdot x_A \geq 1 \cdot x_B$ we obtain

$$W = \begin{pmatrix} 0 & 2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{array}{c} \text{C} \\ \swarrow \quad \searrow \\ \text{A} \quad \xrightarrow{2} \quad \text{B} \\ \swarrow \quad \searrow \\ \text{A} \quad \xrightarrow{1/2} \quad \text{B} \end{array} \quad P = \begin{pmatrix} \cdot & B & \cdot \\ A & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

After processing $1 \cdot x_C \geq 1 \cdot x_B$ and $1 \cdot x_B \geq 1 \cdot x_C$ the procedure yields

$$W = \begin{pmatrix} 0 & 2 & 2 \\ 1/2 & 0 & 1 \\ 1/2 & 1 & 0 \end{pmatrix} \quad \begin{array}{c} \text{C} \\ \swarrow \quad \xrightarrow{1} \quad \searrow \\ \text{A} \quad \xrightarrow{2} \quad \text{B} \\ \swarrow \quad \searrow \\ \text{A} \quad \xrightarrow{1/2} \quad \text{B} \end{array} \quad P = \begin{pmatrix} \cdot & B & B.C \\ A & \cdot & C \\ B.A & B & \cdot \end{pmatrix}$$

Finally, after processing $1 \cdot x_C \geq 1 \cdot x_A$ and $1 \cdot x_A \geq 1 \cdot x_C$ we obtain

$$W = \begin{pmatrix} 2 & 2 & 2 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \end{pmatrix} \quad \begin{array}{c} \text{C} \\ \text{A} \quad \text{B} \end{array} \quad P = \begin{pmatrix} B.C.A & B & B.C \\ C.A & C.A.B & C \\ A & A.B & A.B.C \end{pmatrix}$$


Since some diagonal elements of the first matrix W are greater than one we know that this specification is inconsistent. Moreover, we can identify inconsistent variables and can construct a counter-model: All three variables x_A, x_B, x_C are inconsistent because $W[x_A, x_A]$, $W[x_B, x_B]$ and $W[x_C, x_C]$ are greater than one, respectively. Furthermore, the corresponding elements in matrix P serve as an explanation for the inconsistency. E.g., the path $B.C.A$ starting from A is a witness for the inconsistency of variable x_A .

Thus we have formally shown the inconsistency which we already have observed and discussed informally in Example 4.

Theorem 22 *Given a set I of consistent input inequations, φ_{lb} conditions and $\varphi_{\text{uniq_min}}$ conditions, i.e.,*

$$\begin{aligned} a \cdot x &\leq b \cdot y \\ n &\leq x \\ x > 0 &\implies n \leq y \end{aligned}$$

Algorithm 3 computes the minimal rational solution for all variables x, y in the graph representation of I .

Proof In step 4 by initialising σ with zero we already obtain a solution if we do not consider φ_{lb} and $\varphi_{\text{uniq_min}}$ conditions. Thus we need to consider each φ_{lb} condition:

If a φ_{lb} condition enforces a minimal number of instances we set it, if it is not already fulfilled (steps 5–8). Then we have to update all other variable solutions (steps 9–13). Note that we cannot have update cycles, as we have consistent inequations as input. Thus we propagate values through the graph over maximal weight paths.

Finally we take into account $\varphi_{\text{uniq_min}}$ conditions in steps 14–19. Observe that $\varphi_{\text{uniq_min}}$ conditions are triggered by φ_{lb} conditions. Hence if a $\varphi_{\text{uniq_min}}$ condition becomes active, we add it to the considered φ_{lb} conditions (step 17). \square

The time complexity of Algorithm 3 is $O((|L| + |U|) \cdot (|V| + |U|))$.

Example 23 Let S be the specification $\langle A, \text{mult}, \text{uniq}, \text{lb} \rangle$ depicted in Figure 5.2 consisting of the associations

$$\begin{aligned} a_{AB} &= \{r_{AB} : A, r_{BA} : B\} \\ a_{BC} &= \{r_{BC} : B, r_{CB} : C\} \\ a_{AC} &= \{r_{AC} : A, r_{CA} : C\} \end{aligned}$$

Algorithm 3 Compute a minimal rational solution

Require: Let I be a set of input inequations $a \cdot x \leq b \cdot y$, L the set of φ_{lb} conditions $n \leq x$ and U the set of $\varphi_{\text{uniq_min}}$ conditions $x > 0 \implies n \leq y$

- 1: Let V be the set of variables occurring in I
- 2: Let W be the output $|V| \times |V|$ matrix from Algorithm 2 on I
- 3: Let P be the output $|V| \times |V|$ matrix from Algorithm 2 on I
- 4: Let σ be a mapping from V to the natural numbers, initially mapping all variables to zero
- 5: **for** all φ_{lb} conditions $n \leq x$ in L **do**
- 6: **if** $n > \sigma(x)$ **then**
- 7: $\sigma(x) \leftarrow n$
- 8: **end if**
- 9: **for** all variables y in V such that $x \neq y$ **do**
- 10: **if** $\sigma(x) \cdot W[x, y] > \sigma(y)$ **then**
- 11: $\sigma(y) \leftarrow \sigma(x) \cdot W[x, y]$
- 12: **end if**
- 13: **end for**
- 14: **for** all $\varphi_{\text{uniq_min}}$ conditions $x > 0 \implies n \leq y$ in U **do**
- 15: **if** $\sigma(x) > 0$ **then**
- 16: $U \leftarrow U \setminus \{x > 0 \implies n \leq y\}$
- 17: $L \leftarrow L \cup \{m \leq y\}$
- 18: **end if**
- 19: **end for**
- 20: **end for**
- 21: **return** σ

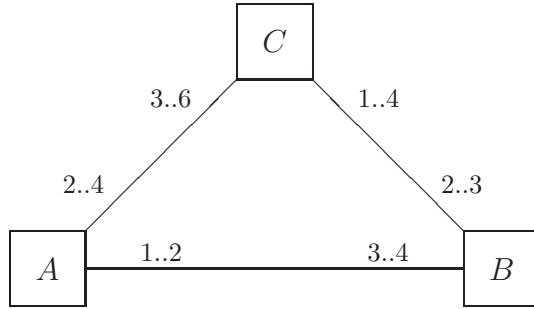


Figure 5.2.: UML specification (nuniq constraints omitted)

with the multiplicities

$$\begin{array}{ll}
 mult(r_{AB}) = [1, 2] & mult(r_{CB}) = [1, 4] \\
 mult(r_{BA}) = [3, 4] & mult(r_{AC}) = [2, 4] \\
 mult(r_{BC}) = [2, 3] & mult(r_{CA}) = [3, 6] ,
 \end{array}$$

the lower bounds

$$\begin{array}{l}
 lb(A) = 1 \\
 lb(B) = 0 \\
 lb(C) = 0 ,
 \end{array}$$

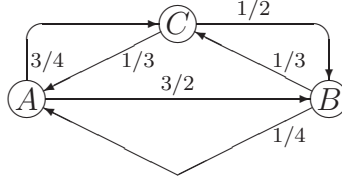
and the uniqueness constraints

$$\begin{array}{ll}
 uniq(r_{AB}) = \text{nuniq} & uniq(r_{CB}) = \text{nuniq} \\
 uniq(r_{BA}) = \text{nuniq} & uniq(r_{AC}) = \text{nuniq} \\
 uniq(r_{BC}) = \text{nuniq} & uniq(r_{CA}) = \text{nuniq} .
 \end{array}$$

The sat-condition $sc(S)$ is the conjunction of

$$\begin{array}{ll}
 x_A \geq 1 & 2 \cdot x_B \geq 3 \cdot x_A \\
 x_B \geq 0 & 4 \cdot x_A \geq 1 \cdot x_B \\
 x_C \geq 0 & 3 \cdot x_C \geq 1 \cdot x_B \\
 & 4 \cdot x_B \geq 2 \cdot x_C \\
 & 4 \cdot x_C \geq 3 \cdot x_A \\
 & 6 \cdot x_A \geq 2 \cdot x_C .
 \end{array}$$

With the maximal-weight-path-algorithm 2 the sat-condition $sc(S)$ gives rise to the following weighted graph



and to following weight matrix

$$W = \begin{pmatrix} 3/8 & 3/2 & 3/4 \\ 1/4 & 3/8 & 1/3 \\ 1/3 & 1/2 & 1/4 \end{pmatrix}$$

As all values in the diagonal of W are strictly smaller than 1 we know that the specification is weakly consistent.

At start of Algorithm 3 we have

$$\sigma(A) = 0 \quad \sigma(B) = 0 \quad \sigma(C) = 0 .$$

We process the first lower bound $x_A \geq 1$ and obtain

$$\sigma(A) = 1 \quad \sigma(B) = \frac{3}{2} \quad \sigma(C) = \frac{3}{4} .$$

Since all other lower bounds are zero there is no need for the algorithm to consider further cases. I.e., we finish with the minimal rational solution

$$\sigma = \begin{pmatrix} 1 \\ 3/2 \\ 3/4 \end{pmatrix} .$$

An integer solution for any variable v can be built by multiplying $\sigma(v)$ by

$$\text{lcm}\{b_x \mid x \text{ occurs in } I\}$$

where b_x denotes the non-negative integer b in each inequation $a \cdot x \leq b \cdot y$. This is a valid operation by Theorem 24.

Theorem 24 *Let I be a system of inequations of the form*

$$a \cdot x \leq b \cdot y ,$$

and let v be some variable. I has a rational solution assigning v a value greater zero if and only if I has a solution over the non-negative integers assigning v a value greater zero.

Proof We show both directions:

(\implies) Trivial, since every solution over the non-negative integers is also a rational solution.

(\impliedby) Let σ be a rational solution such that $\sigma(v) > 0$. Since $\sigma(x)$ is rational for all variables x , $\sigma(x)$ can be written as a_x/b_x for non-negative integers a_x, b_x . Let $n = \text{lcm}\{b_x \mid x \text{ occurs in } I\}$. Then $n \cdot \sigma(x)$ is an integer. Moreover, every multiple of a solution is again a solution, hence $n \cdot \sigma$ is a non-negative integer solution of I . Since $\sigma(v) > 0$ we also have $n \cdot \sigma(v) > 0$. □

Example 25 Reconsider the minimal rational solution for the specification in Example 23:

$$\sigma = \begin{pmatrix} 1 \\ 3/2 \\ 3/4 \end{pmatrix} .$$

For this specification S we will see that the integer solution derived from the rational solution by multiplying with $\text{lcm}\{b_x \mid x \text{ occurs in } I\}$, where b_x denotes the non-negative integer b in each inequation $a \cdot x \leq b \cdot y$, is not necessarily minimal.

Multiplying this rational solution σ with $\text{lcm}(1, 2, 4) = 4$ (this is even a smaller lcm than the general one with the original equation coefficients) yields the solution

$$\tau = 4 \cdot \begin{pmatrix} 1 \\ 3/2 \\ 3/4 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 3 \end{pmatrix} .$$

But now we note that the minimal integer solution π is strictly smaller than τ :

$$\pi = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} < \begin{pmatrix} 4 \\ 6 \\ 3 \end{pmatrix} = \tau .$$

We obtain similar results in this example for the case that we assume all roles to be unique in Figure 5.2, i.e., if

$$\begin{array}{ll} \text{uniq}(r_{AB}) = \text{uniq} & \text{uniq}(r_{CB}) = \text{uniq} \\ \text{uniq}(r_{BA}) = \text{uniq} & \text{uniq}(r_{AC}) = \text{uniq} \\ \text{uniq}(r_{BC}) = \text{uniq} & \text{uniq}(r_{CA}) = \text{uniq} \end{array}$$

would hold. This would give rise to following additional conditions

$$\begin{array}{l} x_A > 0 \implies x_B \geq 3 \\ x_B > 0 \implies x_A \geq 1 \\ x_B > 0 \implies x_C \geq 1 \\ x_C > 0 \implies x_B \geq 2 \\ x_A > 0 \implies x_C \geq 3 \\ x_C > 0 \implies x_A \geq 2 . \end{array}$$

Then the minimal integer solution ρ in this case is

$$\rho = \begin{pmatrix} 2 \\ 3 \\ 3 \end{pmatrix} .$$

So note that in general the lcm solution is not minimal for the unique case either.

Theorem 26 *Given a set I of consistent input inequations, φ_{lb} conditions and $\varphi_{\text{uniq-min}}$ conditions, i.e.,*

$$\begin{aligned} a \cdot x &\leq b \cdot y \\ n &\leq x \\ x > 0 &\implies n \leq y \end{aligned}$$

Algorithm 4 computes the minimal integer solution for all variables x, y in the graph representation of I .

Proof The argumentation is the same as in the proof for Algorithm 3 except that we now cannot be sure that update cycles exist. Nevertheless we can guarantee termination as our input is consistent — thus a solution exists — and the repeat-loop is bounded by Theorem 24 with $\text{lcm}\{b_x \mid x \text{ occurs in } I\}$ iterations, where b_x denotes the non-negative integer b in each inequation $a \cdot x \leq b \cdot y$. \square

Example 27 Recall the specification from Example 23 on page 46, where we computed the minimal rational solution for it. Now we will derive the minimal integer solution with Algorithm 4.

We begin with

$$\sigma(A) = 0 \quad \sigma(B) = 0 \quad \sigma(C) = 0 .$$

Similar to the rational case we process the first lower bound $x_A \geq 1$ and obtain

$$\sigma(A) = 1 \quad \sigma(B) = \left\lceil \frac{3}{2} \right\rceil = 2 \quad \sigma(C) = \left\lceil \frac{3}{4} \right\rceil = 1 .$$

Again, the other lower bounds do not enforce further investigation, and we are finished with the minimal integer solution

$$\sigma = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} .$$

We note that in this example no additional loops are necessary due to rounding up the intermediary rational solutions. Anyway, as we will see, this is not always the case.

In the worst case Algorithm 4 has an exponential time complexity.

Algorithm 4 Compute a minimal integer solution

Require: Let I be a set of input inequations $a \cdot x \leq b \cdot y$, L the set of φ_{lb} conditions $n \leq x$ and U the set of $\varphi_{\text{uniq_min}}$ conditions $x > 0 \implies n \leq y$

- 1: Let V be the set of variables occurring in I
- 2: Let W be the output $|V| \times |V|$ matrix from Algorithm 2 on I
- 3: Let P be the output $|V| \times |V|$ matrix from Algorithm 2 on I
- 4: Let σ be a mapping from $|V|$ variables to natural numbers initialised with zero
- 5: **for** all φ_{lb} conditions $n \leq x$ in L **do**
- 6: **if** $n > \sigma(x)$ **then**
- 7: $\sigma(x) \leftarrow n$
- 8: **end if**
- 9: **repeat**
- 10: **if** $\lceil \sigma(x) \cdot W[x, y] \rceil > \sigma(y)$ **then**
- 11: $\sigma(y) \leftarrow \lceil \sigma(x) \cdot W[x, y] \rceil$
- 12: **end if**
- 13: **until** $\lceil \sigma(x) \cdot W[x, y] \rceil = \sigma(y)$ for all variables x, y such that $x \neq y$
- 14: **for** all $\varphi_{\text{uniq_min}}$ conditions $x > 0 \implies n \leq y$ in U **do**
- 15: **if** $\sigma(x) > 0$ **then**
- 16: $U \leftarrow U \setminus \{x > 0 \implies n \leq y\}$
- 17: $L \leftarrow L \cup \{n \leq y\}$
- 18: **end if**
- 19: **end for**
- 20: **end for**
- 21: **return** σ

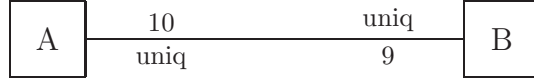


Figure 5.3.: UML class diagram leading to long cycles for Algorithm 4

Remark 28 For the minimal-integer-solution-algorithm 4 we see that in lines 9–13 no iterations build up, if

$$\begin{aligned} m_1 &\leq n_2 \\ n_1 &\leq m_2 \end{aligned}$$

for

$$\begin{aligned} W[x, y] &= \frac{m_1}{n_2} \\ W[y, x] &= \frac{n_1}{m_2} \end{aligned}$$

hold (in case the two classes x, y are connected directly in the specification this means the intervals $[m_1, m_2]$ and $[n_1, n_2]$ overlap) because of

$$\underbrace{\left[\underbrace{\sigma(x) \cdot \frac{m_1}{n_2}}_{\leq \sigma(x)} \cdot \underbrace{\frac{n_1}{m_2}}_{\leq 1} \right]}_{\leq \sigma(x)} \leq \sigma(x) .$$

This means that in this special case Algorithm 4 is polynomial since it behaves similar to Algorithm 3 in this situation.

Example 29 Figure 5.3 shows the specification $\langle A, mult, uniq, lb \rangle$ with $A = \{\{r_1 : A, r_2 : B\}\}$ such that

$$\begin{aligned} mult(r_1) &= [10, 10] \\ mult(r_2) &= [9, 9] \end{aligned}$$

and

$$\begin{aligned} uniq(r_1) &= \text{uniq} \\ uniq(r_2) &= \text{uniq} \end{aligned}$$

hold. This specification leads to many iterations for Algorithm 4 until stabilisation, i.e. until the minimal integer solution is found, assuming we want to have a nontrivial solution (i.e. assume $lb(A) = 1$). We obtain the weight matrix

$$W = \begin{pmatrix} 1 & 9/10 \\ 10/9 & 1 \end{pmatrix},$$

and start iterating with $\sigma(x_A) = 1$:

$\sigma(x_A)$		$\sigma(x_B)$
[1]	= 1	[0.9] = 1
[1.1]	= 2	[1.8] = 2
[2.2]	= 3	[2.7] = 3
[3.3]	= 4	[3.6] = 4
[4.4]	= 5	[4.5] = 5
[5.5]	= 6	[5.4] = 6
[6.6]	= 7	[6.3] = 7
[7.7]	= 8	[7.2] = 8
[8.8]	= 9	[8.1] = 9
[9.9]	= 10	[9] = 9 .

As we see, the simple iterative searching for minimal integer solutions is exponential, as we can make this example parametric: For the multiplicity 10 use 10^k and instead of the multiplicity 9 use the number built by taking k 9s starting with $k = 1$, i.e.,

$$\begin{aligned} & (10, 9) \\ & (100, 99) \\ & (1000, 999) \\ & \vdots \\ & (10^k, \underbrace{9 \dots 9}_{k \text{ times}}) \end{aligned}$$

and so on. Then the iterations are exponential in k .

Example 30 Reconsider the specification from Example 23. In Example 27 we showed how to compute the minimal integer solution, namely

$$\sigma(A) = 1 \quad \sigma(B) = 2 \quad \sigma(C) = 1 .$$

Based upon this result we will now construct a valid configuration with Algorithm 1.

We start with the classes A and B of association $\{r_{AB} : A, r_{BA} : B\}$ and choose a number

$$k \in [\sigma(A) \cdot 3, \sigma(A) \cdot 4] \cap [\sigma(B) \cdot 1, \sigma(B) \cdot 2] ,$$

i.e., $k \in [3, 4] \cap [2, 4] = [3, 4]$. W.l.o.g., we take $k = 3$ and obtain following links

$$\begin{aligned} &(r_{AB} : a_1, r_{BA} : b_1) \\ &(r_{AB} : a_1, r_{BA} : b_2) \\ &(r_{AB} : a_1, r_{BA} : b_1) . \end{aligned}$$

For the classes B and C of association $\{r_{BC} : B, r_{CB} : C\}$ we choose a number

$$k \in [\sigma(B) \cdot 1, \sigma(B) \cdot 4] \cap [\sigma(C) \cdot 2, \sigma(C) \cdot 3] ,$$

i.e., $k \in [2, 8] \cap [2, 3] = [2, 3]$, e.g., $k = 2$. We obtain the links

$$\begin{aligned} &(r_{BC} : b_1, r_{CB} : c_1) \\ &(r_{BC} : b_2, r_{CB} : c_1) . \end{aligned}$$

Finally for the classes A and C of association $\{r_{AC} : A, r_{CA} : C\}$ we choose a number

$$k \in [\sigma(A) \cdot 3, \sigma(A) \cdot 6] \cap [\sigma(C) \cdot 2, \sigma(C) \cdot 4] ,$$

i.e., $k \in [3, 6] \cap [2, 4] = [3, 4]$, e.g., $k = 3$. Then the algorithm returns the links

$$\begin{aligned} &(r_{AC} : a_1, r_{CA} : c_1) \\ &(r_{AC} : a_1, r_{CA} : c_1) \\ &(r_{AC} : a_1, r_{CA} : c_1) . \end{aligned}$$

Now the configuration can be easily constructed by using a_1, b_1, b_2 and c_1 as objects and above computed links as connections between the objects.

6. UML Class Specifications with Multiary Associations

A biologist, a statistician, a mathematician and a computer scientist are on a photo-safari in Africa. As they're driving along the savannah in their jeep, they stop and scout the horizon with their binoculars.

The biologist: "Look! A herd of zebras! And there's a white zebra! Fantastic! We'll be famous!"

The statistician: "Hey, calm down, it's not significant. We only know there's one white zebra."

The mathematician: "Actually, we only know there exists a zebra, which is white on one side."

The computer scientist: "Oh, no! A special case!"

In previous chapters we introduced the concepts consistency and minimality of UML specifications with binary associations. This approach can be generalised to be used with n -ary association where n ranges from 2 to any bigger integer value. Thus we will discuss differences and derive conditions to hold in the generalised framework.

6.1. Extended Linear Diophantine Inequations

We start by describing our observations for a ternary association to get an impression for the different behaviour of n -ary associations ($n \geq 3$) in UML. This special behaviour is due to the semantic interpretation of UML associations in Chen-style (i.e., look-across notation) — recall the discussion in Section 3.2.

Let $a = \{r_1 : c_1, r_2 : c_2, r_3 : c_3\}$ be the ternary association depicted in Figure 6.1 with

$$\begin{aligned} \text{mult}(r_1) &= [L(c_1), U(c_1)] \\ \text{mult}(r_2) &= [L(c_2), U(c_2)] \\ \text{mult}(r_3) &= [L(c_3), U(c_3)] \ . \end{aligned}$$

As in the previous chapter, for each class $c \in Cl$, let x_c be a variable ranging over the non-negative integers. Following equations hold in the symmetric case (where l denotes the number of links):

$$\begin{aligned} x_{c_1} \cdot x_{c_2} \cdot L(c_3) &\leq l \leq x_{c_1} \cdot x_{c_2} \cdot U(c_3) \\ x_{c_1} \cdot L(c_2) \cdot x_{c_3} &\leq l \leq x_{c_1} \cdot U(c_2) \cdot x_{c_3} \\ L(c_1) \cdot x_{c_2} \cdot x_{c_3} &\leq l \leq U(c_1) \cdot x_{c_2} \cdot x_{c_3} \ . \end{aligned}$$

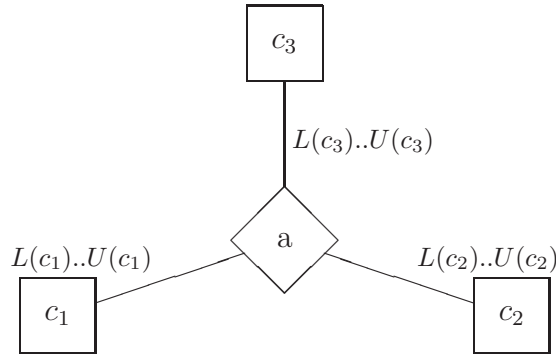


Figure 6.1.: A UML specification with a generic ternary association

They simulate that each combination of objects must have between $L(c)$ and $U(c)$ partner objects of class c .

In addition following conditions must always hold:

$$\begin{aligned} x_{c_1} &\leq x_{c_2} \cdot x_{c_3} \cdot U(c_1) \\ x_{c_2} &\leq x_{c_1} \cdot x_{c_3} \cdot U(c_2) \\ x_{c_3} &\leq x_{c_1} \cdot x_{c_2} \cdot U(c_3) . \end{aligned}$$

These inequations express that no more objects of a distinct class are allowed to be instantiated than possible links can be built by combining all partner objects under full usage of its multiplicity range.

For the symmetric unique case (i.e., $uniq(r_1) = uniq(r_2) = uniq(r_3) = \text{uniq}$) we must also ensure that we have enough distinct objects:

$$\begin{aligned} x_{c_1} > 0 \wedge x_{c_2} > 0 &\implies x_{c_3} \geq L(c_3) \\ x_{c_2} > 0 \wedge x_{c_3} > 0 &\implies x_{c_1} \geq L(c_1) \\ x_{c_1} > 0 \wedge x_{c_3} > 0 &\implies x_{c_2} \geq L(c_2) . \end{aligned}$$

Based upon this insights in the ternary case we can generalise our results from the previous chapters for n -ary associations with $n \geq 2$. Note that we explicitly focus on *symmetric* n -ary associations for several reasons:

- Besides the several possible UML semantic interpretations (which we avoid by formalising definitions from the UML specification) the (intended) semantics of mixed (i.e., asymmetric) n -ary associations remains pretty unclear.
- There is no need for mixed associations in the industry, that means we never saw any mixed multiary association in use (nor the demand for it as conceptual modelling instrument).

- As we will show the efficiency of mixed associations cannot be as optimised as with symmetric associations within our methodology.

So let $\{r_1 : c_1, r_2 : c_2, \dots, r_n : c_n\}$ be a n -ary association with

$$\begin{aligned} mult(r_1) &= [L(c_1), U(c_1)] \\ mult(r_2) &= [L(c_2), U(c_2)] \\ &\vdots \\ mult(r_n) &= [L(c_n), U(c_n)] . \end{aligned}$$

Definition 31 The *satisfiability condition (sat-condition)* for a specification S , denoted by $sc(S)$, is the formula consisting of the conjunction of

$$\bigwedge_{c \in Cl} x_c \geq lb(c)$$

expressing the lower bound constraints, and

$$\bigwedge_{\{\dots, r_i : d, \dots\} \in A} \left[\left(\bigwedge_{\substack{\{\dots, r_j : c, \dots\} \in A \\ c \neq d}} x_c > 0 \wedge x_d = 0 \right) \implies L(d) = 0 \right]$$

expressing that a non-instantiated class type can only appear if the lower multiplicity explicitly allows it, and

$$\left[\left(\bigwedge_{\{\dots, r_i : c, \dots\} \in A} x_c > 0 \right) \implies \bigwedge_{\{\dots, r_i : c, \dots, r_j : d, \dots\} \in A} x_c \cdot L(d) \leq x_d \cdot U(c) \right]$$

expressing the multiplicity constraints similarly to the case with binary associations. In the unique case following conditions

$$\bigwedge_{\{\dots, r_i : d, \dots\} \in A} \left[\left(\bigwedge_{\substack{\{\dots, r_j : c, \dots\} \in A \\ c \neq d}} x_c > 0 \right) \implies x_d \geq L(d) \right]$$

must additionally hold.

These conditions can easily verified by the following lemma stating that our original observation of

$$\begin{aligned} x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot L(c_n) &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot U(c_n) \\ x_{c_1} \cdot x_{c_2} \cdots L(c_{n-1}) \cdot x_{c_n} &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots U(c_{n-1}) \cdot x_{c_n} \\ &\vdots \\ L(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} &\leq l \leq U(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} . \end{aligned} \tag{6.1}$$

is equal to above conditions.

Lemma 32 *The formula*

$$\exists l \bigwedge_{d \in Cl} \left(\prod_{\substack{c \in Cl \\ c \neq d}} x_c \cdot L(d) \leq l \leq \prod_{\substack{c \in Cl \\ c \neq d}} x_c \cdot U(d) \right)$$

is satisfiability-equivalent to

$$\bigwedge_{d, d' \in Cl} \left(\prod_{\substack{c \in Cl \\ c \neq d}} x_c \cdot L(d) \leq \prod_{\substack{c \in Cl \\ c \neq d'}} x_c \cdot U(d') \right)$$

which holds if and only if

$$\begin{aligned} & \left(\exists d, d' : d \neq d' \wedge x_d = 0 \wedge x_{d'} = 0 \right) \\ \vee & \left(\exists d : x_d = 0 \wedge \bigwedge_{d' \neq d} x_{d'} > 0 \wedge L(d) \leq U(d) \wedge L(d) = 0 \right) \\ \vee & \left(\bigwedge_{c \in Cl} x_c > 0 \wedge \bigwedge_{c, d \in Cl} x_c \cdot L(d) \leq x_d \cdot U(c) \right) \end{aligned}$$

holds.

Further we note that the inequations

$$\begin{aligned} x_{c_1} &\leq x_{c_2} \cdots x_{c_n} \cdot U(c_1) \\ &\vdots \\ x_{c_n} &\leq x_{c_1} \cdots x_{c_{n-1}} \cdot U(c_n) \end{aligned}$$

which we observed in the example are subsumed by Equations 6.1.

An Excursion to Balanced Sequences of Tuples An initial subsequence $\langle t_1, \dots, t_k \rangle$ of a sequence $\tau = \langle t_1, \dots, t_k, \dots \rangle$ is called a prefix of τ . A sequence τ of k elements over a set T is called *evenly distributed* w.r.t. T , if every element of T occurs at least $\lfloor k/|T| \rfloor$ and at most $\lceil k/|T| \rceil$ times in τ .

Let $t|_j$ denote the tuple obtained from $t = (t_1, \dots, t_n)$ by removing the j -th component:

$$t|_j = (t_1, \dots, t_{j-1}, t_{j+1}, \dots, t_n) .$$

The operation is extended homomorphically to sets and sequences of tuples:

$$\begin{aligned} T|_j &= \{t|_j \mid t \in T\} \\ \langle t_1, t_2, t_3, \dots \rangle|_j &= \langle t_1|_j, t_2|_j, t_3|_j, \dots \rangle . \end{aligned}$$

We call a sequence τ of n -tuples over a set T *balanced* w.r.t. T if for all $1 \leq j \leq n$, the sequence $\tau|_j$ is evenly distributed w.r.t. $T|_j$.

Let $\mathbb{Z}_i = \{0, \dots, i-1\}$.

Conjecture 33 Let i_1, \dots, i_n be any positive integers. There is a sequence τ enumerating the elements of $T = \mathbb{Z}_{i_1} \times \dots \times \mathbb{Z}_{i_n}$ such that every prefix of τ is balanced w.r.t. T .

Example 34 The following enumeration of $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_3$ satisfies the conjecture:

(0, 0, 0)
(1, 1, 0)
(0, 1, 2)
(1, 0, 1)
(0, 1, 1)
(0, 0, 2)
(1, 1, 2)
(1, 0, 0)
(0, 0, 1)
(1, 1, 1)
(0, 1, 0)
(1, 0, 2)

Example 35 Let i_1, \dots, i_n be pairwise coprime. The sequence over $\mathbb{Z}_{i_1} \times \dots \times \mathbb{Z}_{i_n}$ that starts with $(0, \dots, 0)$ and continues by adding 1 (modulo i_j) simultaneously to every component of the preceding element, satisfies the conjecture.

Example 36 The following sequences over \mathbb{Z}_2^n satisfy the conjecture:

00 000 000 0000 0000
11 110 011 1100 0011
01 011 101 0110 0101
10 101 110 1010 0110
001 001 0011 1001
111 010 1111 1010
010 100 0101 1100
100 111 1001 1111
0001 0001
1101 0010
0111 0100
1011 0111
0010 1000
1110 1011
0100 1101
1000 1110

Let $sc(S)$ be the conjunction of above constraints for a given specification S .

Theorem 37 A specification S is weakly consistent if and only if the formula $sc(S)$ is solvable.

The proof of the theorem assumes the validity of Conjecture 33.

Proof We prove the two directions separately.

(\implies) Let the specification $S = \langle A, mult, uniq, lb \rangle$ be weakly consistent, and let $C = \langle O, L, class, ass, obj \rangle$ be a configuration satisfying S . W.l.o.g. we assume that C is normalised, i.e., that it does not contain multiple links corresponding to $uniq$ - $uniq$ associations.

The lower bound constraints $x_c \geq lb(c)$ hold since we have $|c| \geq lb(c)$ for all classes c .

For the symmetric case (i.e., $uniq(r_1) = \dots = uniq(r_n)$) now consider n classes c_1, c_2, \dots, c_n connected by a n -ary association $\{r_1 : c_1, r_2 : c_2, \dots, r_n : c_n\}$. Let l be the number of all links connecting $n - 1$ objects with the remaining object. Because of

$$\begin{aligned} \delta(\{r_1 : c_1, \dots, r_{n-1} : c_{n-1}\}) &\in mult(r_n) = [L(c_n), U(c_n)] \\ \delta(\{r_1 : c_1, \dots, r_{n-2} : c_{n-2}, r_n : c_n\}) &\in mult(r_{n-1}) = [L(c_{n-1}), U(c_{n-1})] \\ &\vdots \\ \delta(\{r_2 : c_2, \dots, r_n : c_n\}) &\in mult(r_1) = [L(c_1), U(c_1)] \end{aligned} \quad (6.2)$$

we obtain for the first condition with $n - 1$ fixed objects of class $c_1 \dots c_{n-1}$ the inequation

$$L(c_n) \leq l_{c_1 \dots c_{n-1}} \leq U(c_n) \text{ ,}$$

where $l_{c_1 \dots c_{n-1}}$ denotes the number of links between the tuple of $n - 1$ fixed objects and the remaining object of class c_n . This generalises to following inequation

$$x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot L(c_n) \leq l \leq x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot U(c_n) \text{ ,}$$

since it expresses all possible combinations of objects of classes $c_1 \dots c_{n-1}$.

With the same arguments we build constraints for the remaining $n - 1$ combinations of Condition 6.2 and obtain

$$\begin{aligned} x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot L(c_n) &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot U(c_n) \\ x_{c_1} \cdot x_{c_2} \cdots L(c_{n-1}) \cdot x_{c_n} &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots U(c_{n-1}) \cdot x_{c_n} \\ &\vdots \\ L(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} &\leq l \leq U(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} \end{aligned} \quad (6.3)$$

Finally, if $uniq(r_1) = uniq$ (similar argumentation for $uniq(r_i)$ with $i = 2..n$) holds then any $n - 1$ partner objects of class $c_2 \dots c_n$ have to be connected to at least $L(c_1)$ different objects of class c_1 , i.e., if the number of partner objects is greater than zero, then the number of c_1 -objects has to be greater than $L(c_1)$:

$$x_{c_2} > 0 \wedge \dots \wedge x_{c_n} > 0 \implies x_{c_1} \geq L(c_1) \text{ .}$$

(\Leftarrow) Given a specification S and a solution σ of the sat-condition $sc(S)$ we construct a configuration $C = \langle O, L, class, ass, obj \rangle$ that satisfies S .

For all classes $c \in Cl$, we define O and $class$ such that O contains exactly $\sigma(c)$ objects of class c . We construct L and obj according to Algorithm 5.

The number chosen in line 4 exists by definition of $sc(S)$ and Lemma 32. The sequence computed in line 5 exists by Conjecture 33. Since the sequence fulfils the condition that every prefix is balanced w.r.t. T , we know that the links follow a uniform distribution for n -ary associations. That means we obtain in the non-unique case

$$\left\lfloor l / \prod_{\substack{d \in Cl \\ c \neq d}} \sigma(d) \right\rfloor \leq \delta(obj(e) \setminus \{r_c : o\}) \leq \left\lceil l / \prod_{\substack{d \in Cl \\ c \neq d}} \sigma(d) \right\rceil, \quad (6.4)$$

where $obj(e)$ denotes the components of any edge e . By the choice of l we have

$$\prod_{\substack{d \in Cl \\ c \neq d}} x_d \cdot L(c) \leq l \leq \prod_{\substack{d \in Cl \\ c \neq d}} x_d \cdot U(c)$$

and we obtain

$$L(c) \leq \delta(obj(e) \setminus \{r_c : o\}) \leq U(c)$$

by simple arithmetic.

A similar argumentation holds for the symmetric unique case. Since we may assume a uniform distribution we can be sure that the links in Equation 6.4 have different partner objects. Therefore we conclude

$$L(c) \leq \gamma_{r_c}(obj(e) \setminus \{r_c : o\}) \leq U(c)$$

with the same rationale as in the non-unique case.

We conclude that the configuration C satisfies the specification S . \square

Example 38 Let $a = \{r_1 : A, r_2 : B, r_3 : C, r_4 : D\}$ be the 4-ary association of the specification S depicted in Figure 6.2 with

$$\begin{aligned} mult(r_1) &= [1, 2] & mult(r_2) &= [1, 2] \\ mult(r_3) &= [2, 4] & mult(r_4) &= [1, 3] \end{aligned}$$

and

$$\begin{aligned} uniq(r_1) &= \text{uniq} & uniq(r_2) &= \text{uniq} \\ uniq(r_3) &= \text{uniq} & uniq(r_4) &= \text{uniq} . \end{aligned}$$

Further assume

$$lb(A) = lb(B) = lb(C) = lb(D) = 1 ,$$

Algorithm 5 Constructing the links of a configuration with multiary associations

- 1: let $L = \emptyset$
- 2: let $\mathbb{Z}_{i_j} = \{0, \dots, i - 1\}$ of the j -th class where j has i instantiations
- 3: **for** all sets $\{c_1, \dots, c_n\} \in 2^{C_l}$ with $\{r_1 : c_1, \dots, r_n : c_n\} \in A$ **do**
- 4: choose a number l such that

$$\begin{aligned} x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot L(c_n) &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot U(c_n) \\ x_{c_1} \cdot x_{c_2} \cdots L(c_{n-1}) \cdot x_{c_n} &\leq l \leq x_{c_1} \cdot x_{c_2} \cdots U(c_{n-1}) \cdot x_{c_n} \\ &\vdots \\ L(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} &\leq l \leq U(c_1) \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot x_{c_n} \end{aligned}$$

holds.

- 5: Compute a sequence τ enumerating the elements of

$$T = \mathbb{Z}_{i_1} \times \cdots \times \mathbb{Z}_{i_n}$$

(matching the domains of each class) such that every prefix of τ is balanced with restriction to T

- 6: Build links with matching roles r_1, \dots, r_n and objects corresponding with the indices as proposed by τ
 - 7: Add these links to L
 - 8: **end for**
 - 9: **return** L
-

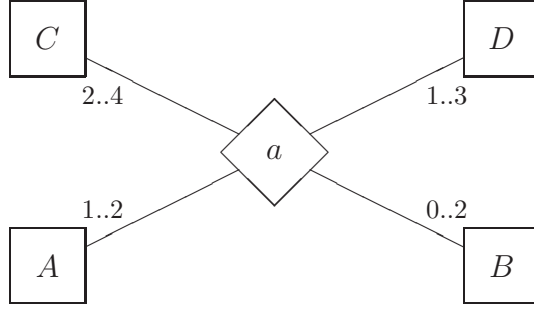


Figure 6.2.: A 4-ary association (without drawn uniq constraints)

i.e., we want to avoid the trivial empty solution.

Then the sat-condition $sc(S)$ is the conjunction of

$$\begin{array}{ll} x_A \geq 1 & x_B \geq 1 \\ x_C \geq 1 & x_D \geq 1 \end{array}$$

and

$$\begin{array}{ll} x_C \cdot 1 \leq x_D \cdot 4 & x_D \cdot 2 \leq x_C \cdot 3 \\ x_B \cdot 1 \leq x_D \cdot 2 & x_B \cdot 2 \leq x_C \cdot 2 \\ x_A \cdot 1 \leq x_D \cdot 2 & x_A \cdot 2 \leq x_C \cdot 2 \end{array}$$

and

$$\begin{array}{ll} x_D \cdot 0 \leq x_B \cdot 3 & x_D \cdot 1 \leq x_A \cdot 3 \\ x_C \cdot 0 \leq x_B \cdot 4 & x_C \cdot 1 \leq x_A \cdot 4 \\ x_A \cdot 0 \leq x_B \cdot 2 & x_B \cdot 1 \leq x_A \cdot 2 \end{array}$$

and

$$\begin{array}{ll} x_A \geq 1 & x_B \geq 0 \\ x_C \geq 2 & x_D \geq 1 \end{array} .$$

Note that some subsumed constraints which evaluate always to true under the other stated constraints have already been removed in order to simplify the presented output.

Example 39 In this example we see that in general our presented conditions cannot be applied for asymmetric n -ary associations. So let $a = \{r_1 : A, r_2 : B, r_3 : C\}$ be the

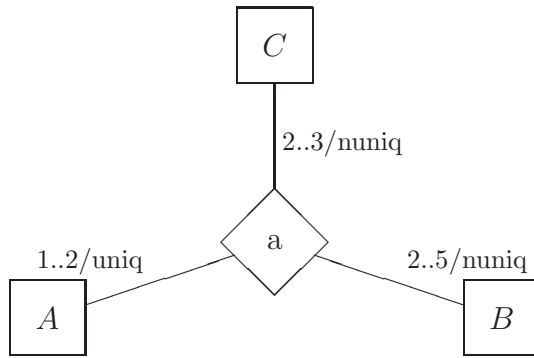


Figure 6.3.: A UML specification with an asymmetric ternary association

ternary association of the specification S depicted in Figure 6.3 with

$$\begin{aligned} mult(r_1) &= [1, 2] \\ mult(r_2) &= [2, 5] \\ mult(r_3) &= [2, 3] \end{aligned}$$

and

$$\begin{aligned} uniq(r_1) &= \text{uniq} \\ uniq(r_2) &= \text{nuniq} \\ uniq(r_3) &= \text{nuniq} . \end{aligned}$$

Further assume

$$lb(A) \geq 1 .$$

We obtain

$$\begin{aligned} x_A \cdot x_B \cdot 2 &\leq l \leq x_A \cdot x_B \cdot 3 \\ x_A \cdot 2 \cdot x_C &\leq l \leq x_A \cdot 5 \cdot x_C \\ 1 \cdot x_B \cdot x_C &\leq l \leq 2 \cdot x_B \cdot x_C \end{aligned}$$

and

$$\begin{aligned} x_A &\leq x_B \cdot x_C \cdot 2 \\ x_B &\leq x_A \cdot x_C \cdot 5 \\ x_C &\leq x_A \cdot x_B \cdot 3 \end{aligned}$$

and

$$x_A \geq 1 .$$

Now consider the configuration with $x_A = 1$, $x_B = 2$, and $x_C = 1$ consisting of five links:

$$\begin{aligned} &(a_1, b_1, c_1) \\ &(a_1, b_1, c_1) \\ &(a_1, b_1, c_1) \\ &(a_1, b_2, c_1) \\ &(a_1, b_2, c_1) \end{aligned}$$

This is a valid configuration according to the definition of UML associations, but our inequations deny the usage of five links (i.e., $l = 5$) because

$$1 \cdot 2 \cdot 1 \leq 5 \not\leq 2 \cdot 2 \cdot 1 .$$

Even neglecting the relevant constraint, i.e.,

$$1 \cdot x_B \cdot x_C \leq l [\leq 2 \cdot x_B \cdot x_C]$$

does not lead to a valid general approach, since then the configuration

$$\begin{array}{ll} (a_1, b_1, c_1) & (a_1, b_1, c_1) \\ (a_1, b_2, c_1) & (a_1, b_2, c_1) \\ (a_2, b_1, c_1) & (a_2, b_1, c_1) \\ (a_2, b_2, c_1) & (a_2, b_2, c_1) \\ (a_3, b_1, c_1) & (a_3, b_1, c_1) \\ (a_3, b_2, c_1) & (a_3, b_2, c_1) \end{array}$$

with $x_A = 3$, $x_B = 2$, $x_C = 1$ and $l = 12$ is accepted by the inequations but does not form a valid configuration according to the UML definition.

6.2. Solving Extended Linear Diophantine Inequations

As we already know solving Diophantine inequations without further knowledge about their structure often leads to runtime performance in NP. Anyway, in the previous chapter we have presented an efficient approach for solving our Diophantine inequations by taking advantage of their variable arrangements. This led us to a weighted directed graph which traversal told us valuable information on the system of inequations consistency.

In this section we will discuss the new extended linear Diophantine inequations in the n -ary case and show that by the reduction stated in Lemma 32 we can use our efficient methods as presented in the binary case. That means although we originally have up to $n - 1$ variables in a single inequation, e.g.,

$$x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot L(c_n) \leq l \leq x_{c_1} \cdot x_{c_2} \cdots x_{c_{n-1}} \cdot U(c_n) ,$$

Lemma 32 shows the transformation into inequations with only two variables by pairwise combination of all multi-variable inequations. The conditions introduced by this

transformation for the special case with the lower multiplicity being zero can also be easily checked, since they need only be checked once for a given consistency problem, i.e., whether e.g.,

$$x_{c_1} > 0 \wedge x_{c_2} > 0 \wedge \dots \wedge x_{c_n} = 0 \implies L(c_n) = 0 \quad (6.5)$$

holds. As in the binary case lower bound constraints and uniqueness constraints need not be considered for consistency since a solution satisfying these constraints can always be found by multiplication of the checked solution.

In the same manner the algorithms for constructing minimal models hold for our approach in the extended n -ary case. Since the $sc(S)$ conditions state either only inequations with two variables or easily checkable conditions as Equation 6.5 both the algorithms for finding minimal rational as for finding minimal integer solution can be used without modification.

Example 40 Have a look at the inequations from Example 38. Although the $sc(S)$ conditions have to deal with a specification S consisting of a 4-ary associations we obtain only linear Diophantine inequations with at most two variables, i.e., of the form

$$x_c \geq n$$

with $n \in \mathbb{N}$ and

$$x_c \cdot L(d) \leq x_d \cdot U(c) .$$

As a result all existing methodology from previous chapters can be fully applied.

Thus a major contribution can be seen in handling consistency checking and minimality computation for symmetric n -ary associations with the same efficiency as with binary associations under the UML semantics (i.e., look-across semantics) with its special behaviour for n -ary associations (see the detailed discussion in Section 3.2). This is an extension to the results already known in the literature for associations under look-here semantics.

7. Incremental Update of UML Class Configurations

*It would be illogical to assume that all conditions remain stable.
Spock, "The Enterprise Incident", stardate 5027.3*

As a final consideration in this thesis we consider the problem of changing demands in configuration management. That means we investigate the situation for configurations with changing requirements after we have computed minimal solutions and have checked consistency. Consider for example a system of network switches which need to be extended to fulfil higher network performance demands: normally you would like to add a few switches without restructuring the whole network.

7.1. Different Requirements for Changing Configurations

In order to formalise incremental configuration updates we need to distinguish different requirements regarding extensibility. We can identify at least three approaches:

1. The specification changes whereas the configuration stays unchanged. This means that without new direct configuration-specific requirements the configuration can easily become inconsistent. Under this circumstance we do not think an incremental approach is a good solution and prefer to solve the (possibly completely new) problem with our existing framework presented in the previous chapters.
2. The specification stays unchanged and only new requirements are set regarding the given valid configuration. We restrict the new requirements to adding new instances and links (i.e., a real extension) but forbid the reduction of existing instances. That means new constraints consist only of (higher than existing) lower bound requirements $lb(c) \geq n$ for $n \in \mathbb{N}$. Further we work under the assumption that links between instances are cheap. Hence the major minimality criterion is a class-based minimality argument.
3. The specification stays unchanged and only new requirements are set regarding the given valid configuration. Similar to the previous approach we only consider real extensions but under the assumption that links are also an important aspect for minimality aspects. This means that link creation can be considered as an expensive operation.

7.2. Strategies for Handling Incremental Updates

We restrict our considerations to symmetric associations with the same arguments as for n -ary associations in the previous chapter. Under this assumption we are going to present strategies for handling the two approaches we identified in the previous section for extending configurations.

Extending Configurations under Class-Based Minimality

Let us at first consider the situation for symmetric associations in the non-unique case. Then the following procedure describes a valid strategy for solving the problem:

- 1: Let L be an empty set of links
- 2: Let $S = \langle A, mult, uniq, lb \rangle$ be the input specification
- 3: Let C be the valid configuration to be extended
- 4: Let lb' denote the new lower bound constraints, and let $S' = \langle A, mult, uniq, lb' \rangle$
- 5: Solve $sc(S')$ with our existing framework, i.e., we know the number of necessary instances for each class and the number of maximal necessary links l . Note that we take the maximum number of allowed links since links are inexpensive and the maximum guarantees us that no instance will have to few links as long as the input configuration is valid
- 6: Insert the existing links from C into the new configuration C' , i.e., to L
- 7: Build the remaining $l - l_{old}$ links by choosing the objects with the minimal number of connected links at each side of the association and add them to L
- 8: **return** L

The correctness of the algorithm can be verified since we have proven in previous chapters that our framework calculates the correct number of links and that we can find a configuration for the given amount of links. Since the non-unique case does not enforce any constraints on the link arrangement (e.g., no doubled links) we are done.

When dealing with the unique case the situation gets more problematic. But we take advantage of the fact that links are (very) cheap (e.g., imagine a component system of network switches: it is far cheaper and easier to rearrange some cables than buying more hardware switches than actually necessary):

- 1: Let $L = \emptyset$
- 2: Let $S = \langle A, mult, uniq, lb \rangle$ be the input specification
- 3: Let C be the valid configuration to be extended
- 4: Let lb' denote the new lower bound constraints, and let $S' = \langle A, mult, uniq, lb' \rangle$
- 5: Solve $sc(S')$ with our existing framework
- 6: If the existing links are not absolutely uniformly distributed and do not exceed the minimal number of necessary links at each object, simply remove all existing links
- 7: Perform our existing link algorithms either on the (well-arranged) existing links or start with the empty graph.
- 8: Add these created links to L

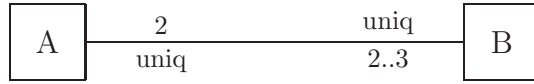


Figure 7.1.: A UML specification in the context of extending configurations

9: **return** L

Since we either start with the empty graph (where everything has been shown in previous chapters to work) or with well-arranged links (which represent an intermediary stop within the original algorithm by definition) we are done, too.

Extending Configurations under Class- and Link-Based Minimality

The task of extending configurations when both class instances and link instances should be kept minimal shows to be a complex undertaking. This results in a numerical optimisation problem to be solved:

- 1: Let $L = \emptyset$
- 2: Let $S = \langle A, mult, uniq, lb \rangle$ be the input specification
- 3: Let C be the valid configuration to be extended
- 4: Let lb' denote the new lower bound constraints, and let $S' = \langle A, mult, uniq, lb' \rangle$
- 5: Solve $sc(S')$ with our existing framework, i.e., we know the number of necessary instances for each class and the number of maximal necessary links l .
- 6: Compute the solution to the numerical optimisation problem of finding the minimal number of link relocations such that both the number of instances and of links is minimal
- 7: Build these links and add them to L
- 8: **return** L

The algorithm is correct by definition of the optimisation problem. The complexity of this task is at least in NP, naive implementations will show exponential runtime behaviour due to backtracking of possible link relocation combinations. It seems to be clear that the non-unique case should be handled much easier in average cases since fewer constraints are active whereas unique constraints tend to trigger more link deletions (since doubled links must be avoided as long as other combinations are possible).

Example 41 In this example we can see a simple extension of our existing framework with systems of linear Diophantine inequations that does in general not suffice to solve the problem of extending configurations efficiently. Anyway, it presents a possible approach and shows which solutions become possible due to the interaction between the existing configuration and the newly created objects and links.

Let S be the specification with the association $\{r_1 : A, r_2 : B\}$ depicted in Figure 7.1

such that

$$\begin{aligned} mult(r_1) &= [2, 2] \\ mult(r_2) &= [2, 3] \end{aligned}$$

and

$$\begin{aligned} uniq(r_1) &= \text{uniq} \\ uniq(r_2) &= \text{uniq} \end{aligned}$$

hold. Assume the valid configuration with $x_A = 3$, $x_B = 4$ and $l = 8$ consisting of the following links

$$\begin{array}{ll} (r_1 : a_1, r_2 : b_1) & (r_1 : a_2, r_2 : b_1) \\ (r_1 : a_1, r_2 : b_2) & (r_1 : a_2, r_2 : b_2) \\ (r_1 : a_1, r_2 : b_3) & (r_1 : a_2, r_2 : b_4) \\ (r_1 : a_3, r_2 : b_3) & (r_1 : a_3, r_2 : b_4) \end{array}$$

is given. Further assume the extension constraint enforces an additional A instance, i.e., $lb(A) \geq 4$. Then one might have the idea to extend our original inequations to

$$\begin{aligned} 2 \cdot x'_A &\leq l' \leq 3 \cdot x'_A + 1 \\ 2 \cdot x'_B &\leq l' \leq 2 \cdot x'_B \ , \end{aligned}$$

where the primed variables denote the number of instances that are needed in addition to the existing ones and where the coefficients leading to inhomogeneous inequations denote the number of free links for objects of that class. Then the inequation simplifies to

$$2 \cdot x'_a \leq 2 \cdot x'_b \leq 3 \cdot x'_a + 1 \ .$$

Now consider the solutions for this equation:

x'_A	x'_B
1	1
1	2
2	2
2	3
3	3
3	4
3	5

The last line, i.e., a configuration with 3 new A objects and 5 new B objects is a case which becomes only possible under the already existing configuration since the one remaining attachment point at a_3 is needed.

8. Conclusion

Beware of computerized fortune-tellers!

In this thesis we presented efficient methods for configuration management, i.e., methods for consistency checking or for the construction of minimal solutions for configuration systems. Due to the wide acceptance of UML as modelling instrument in industry we chose to use UML as our specification language. We started with an overview of existing approaches especially in the ER literature and described the differences between ER and UML semantics under different interpretations. We gave a formal semantic definition for relevant UML parts, i.e., class diagrams with selected association types, like binary and n -ary associations with uniqueness constraints. We presented methods for checking the consistency of specifications that work in polynomial time. This is done by a translation of UML specifications to systems of linear Diophantine inequations. We proposed algorithms to solve these inequations efficiently and gave strategies for constructing configurations from the solutions to these inequations. For the minimality computations we introduced algorithms based on weighted directed graphs for which intelligent traversal returns the minimal rational solution in polynomial time. These results were shown to hold both for binary associations and for the generalised case with n -ary associations. We gave an outlook on the situation for incrementally updating given configurations under new constraints. We identified several possible demands in this context of extending configurations and proposed strategies to solve each of them. Although several important problems have been solved in this thesis there is much room for further research in this area. For example one could investigate the situation for incremental updates more thoroughly and come up with ideas for more efficient approaches. Another issue might be the enumeration of all possible solutions, e.g., as linear combination of base vectors, and not only the minimal solutions.

A. Resources

Human resources are human first, and resources second.
J. Garbers

This PhD thesis is online available at <http://www.logic.at/staff/feinerer>.

The material consists of

- the PhD thesis in PDF format,
- the PhD thesis in Ps format,
- an archive containing implemented algorithms and prototypes. Available are
 - a C program to build inequations from UML diagrams in XML metadata interchange (XMI) format,
 - a Prolog program to check δ - and γ -conditions, to compute links for a specification, and to compute a uniform link distribution, and
 - several R functions implementing the algorithms for solving Diophantine equations by [Ajili and Contejean \[1997\]](#).
- a BibTeX entry to be used for citing this thesis.

```
@PhdThesis{Feinerer_TU_2007,  
  author =      {Ingo Feinerer},  
  title =      {A Formal Treatment of UML Class Diagrams as an  
                Efficient Method for Configuration Management},  
  school =     {Theory and Logic Group, Institute of Computer  
                Languages, Vienna University of Technology},  
  year =       2007,  
  month =      {March}  
}
```

Bibliography

- Farid Ajili and Evelyne Contejean. Avoiding slack variables in the solving of linear diophantine equations and inequations. *Theoretical Computer Science*, 173(1):183–208, 1997. ISSN 0304-3975. doi: [http://dx.doi.org/10.1016/S0304-3975\(96\)00195-8](http://dx.doi.org/10.1016/S0304-3975(96)00195-8).
- Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- R. E. Bellmann. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
- Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams using description logic based systems. In *Proceedings of the KI'2001 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, <http://ceur-ws.org/Vol-44/>, 2001.
- Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1–2):70–118, July 2005. ISSN 0004-3702. URL <http://www.inf.unibz.it/~calvanese/papers-html/AIJ-2005.html>.
- Diego Calvanese and Maurizio Lenzerini. On the interaction between isa and cardinality constraints. In *Proceedings of the 10th IEEE International Conference on Data Engineering (ICDE'94)*, pages 204–213. IEEE Computer Society Press, 1994.
- Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class based representation formalisms. In *Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 109–120. Morgan Kaufmann, 1994.
- Peter Pin-Shan Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions Database Systems*, 1(1):9–36, 1976. ISSN 0362-5915. doi: <http://doi.acm.org/10.1145/320434.320440>.
- Michael Clausen and Albrecht Fortenbacher. Efficient solution of linear diophantine equations. *Journal of Symbolic Computation*, 8(1-2):201–216, 1989. ISSN 0747-7171.
- Evelyne Contejean and Hervé Devie. An efficient incremental algorithm for solving systems of linear diophantine equations. *Information and Computation*, 113(1):143–172, 1994. ISSN 0890-5401. doi: <http://dx.doi.org/10.1006/inco.1994.1067>.

- Zinovy Diskin and Juergen Dingel. Mappings, maps and tables: Towards formal semantics for associations in uml2. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems (Proceedings of the 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006)*, volume 4199 of *Lecture Notes in Computer Science*, pages 230–244. Springer-Verlag, November 2006. ISBN 978-3-540-45772-5. doi: 10.1007/11880240.
- Konrad Engel and Sven Hartmann. Constructing realizers of semantic entity relationship schemes. Technical report, Universität Rostock, Fachbereich Mathematik, 18051 Rostock, Germany, 1995.
- Andreas Falkner and Gerhard Fleischanderl. Configuration requirements from railway interlocking stations. In *IJCAI-01 Workshop on Configuration*, Seattle, WA, 2001.
- Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner, and Markus Zanker. Uml as knowledge acquisition frontend for semantic web configuration knowledge bases. In *RuleML 2002, Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, 14 June 2002, Sardinia, Italy (In conjunction with the First International Semantic Web Conference ISWC2002 and hosted by SIG2 of the OntoWeb Network)*, volume 60 of *CEUR Workshop Proceedings*, 2002a.
- Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration knowledge representation using UML/OCL. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 49–62, London, UK, 2002b. Springer-Verlag. ISBN 3-540-44254-5.
- Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison Wesley Professional, 3rd edition, September 2003. ISBN 0321193687.
- Gonzalo Génova, Juan Llorens, and Paloma Martínez. Semantics of the minimum multiplicity in ternary associations in uml. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 329–341, London, UK, 2001. Springer-Verlag. ISBN 3-540-42667-1.
- Gonzalo Génova, Juan Llorens, and Paloma Martínez. The meaning of multiplicity of n-ary associations in uml. *Software and System Modeling*, 1(2):86–97, 2002.
- Sven Hartmann. Reasoning about participation constraints and chen’s constraints. In Klaus-Dieter Schewe and Xiaofang Zhou, editors, *Database Technologies 2003, Proceedings of the 14th Australasian Database Conference, ADC 2003, Adelaide, South Australia, February 2003*, volume 17 of *CRPIT*, pages 105–113. Australian Computer Society, 2003.
- Martin Hitz and Gerti Kappel. *UML @ Work*. dpunkt.verlag, 2nd edition, 2003. ISBN 3-89864-194-5.

- Trevor Jones and Il-Yeol Song. Analysis of binary/ternary cardinality combinations in entity-relationship modeling. *Data and Knowledge Engineering*, 19(1):39–64, 1996. ISSN 0169-023X. doi: [http://dx.doi.org/10.1016/0169-023X\(95\)00036-R](http://dx.doi.org/10.1016/0169-023X(95)00036-R).
- Trevor Jones and Il-Yeol Song. Binary equivalents of ternary relationships in entity-relationship modeling: a logical decomposition approach. *Journal of Database Management*, 11(2):12–19, 2000. ISSN 1063-8016.
- Jeffrey Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985. ISSN 0097-5397. doi: <http://dx.doi.org/10.1137/0214016>.
- Maurizio Lenzerini and Paolo Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990. ISSN 0306-4379. doi: [http://dx.doi.org/10.1016/0306-4379\(90\)90048-T](http://dx.doi.org/10.1016/0306-4379(90)90048-T).
- Object Management Group. *Meta-Object Facility Specification*, April 2002. URL <http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf>. [Online; accessed 3-January-2007].
- Object Management Group. *UML 2.0 Object Constraint Language Specification*, May 2006. URL <http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf>. [Online; accessed 3-January-2007].
- Object Management Group. *Unified Modeling Language 2.0 Superstructure Specification*, 2005. URL <http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf>. [Online; accessed 3-January-2007].
- Arnold Rochfeld. Merise, an information system design and development methodology, tutorial. In Stefano Spaccapietra, editor, *Entity-Relationship Approach: Ten Years of Experience in Information Modeling, Proceedings of the Fifth International Conference on Entity-Relationship Approach, Dijon, France, November 17-19, 1986*, pages 489–528. North-Holland, 1986. ISBN 0-444-70255-5.
- Arnold Rochfeld and Hubert Tardieu. MERISE: An information system design and development methodology. *Information and Management*, 6:143–159, 1983.
- Gottfried Schenner and Andreas Falkner. Ideas for removing constraint violations with heuristic repair. In *Working Notes of Configuration – Papers from the Workshop at ECAI-2002*, Lyon, France, July 2002.
- Gottfried Schenner and Gerhard Fleischanderl. Modifying configurations with model finding. In *Proceedings of the International Joint Conferences on Artificial Intelligence 2003*, Acapulco, Mexico, 2003. Morgan Kaufmann.
- Perdita Stevens. On the interpretation of binary associations in the unified modelling language. *Software and Systems Modeling*, 1(1):68–79, September 2002. doi: 10.1007/s10270-002-0002-x.

The Alliance for Telecommunications Industry Solutions. Atis telecom glossary 2000, 2000. URL <http://www.atis.org/>. Approved February 28, 2001 by the American National Standards Institute (ANSI).

Index

- sc, *see* Satisfiability condition
- δ , 31
- γ , 31
- Actual tuples, 27
- Association, 19
- Attribute, 19
- Balanced, 59
- Cardinality, 31
- Cardinality ratio constraint, 14
- Class, 19
- Class diagram, 19
- Configuration, 30
- Entity, 12
- Entity Relationship Diagram, 12
- ER, *see* Entity Relationship Diagram
- Inconsistent, 31
- Incremental, 68
- Interpretation, 35
- Limping links, 27
- Look-across, 23
- Look-here, 23
- Mapping, 12
- Method, 19
- Minimal integer solution, 51
- Minimal rational solution, 46
- Multiplicity, 21
- Non-unique, 21
- Normalised, 33
- Object Constraint Language, 21
- OCL, *see* Object Constraint Language
- Potential tuples, 27
- Realizer, 17
- Relationship, 12
- Role, 12
- Satisfiability, 31
- Satisfiability condition, 35
- Sequence, 59
- SERM schema, 13
- Solution, 35
- Specification, 29
- Strongly consistent, 31
- Symmetric, 30
- UML, *see* Unified Modeling Language
- UML-constraint, 41
- Unified Modeling Language, 19
- Unique, 21
- Uniqueness constraint, 21
- Weakly consistent, 31

