

Consistency and Minimality of UML Class Specifications with Multiplicities and Uniqueness Constraints

Ingo Feinerer Gernot Salzer
Technische Universität Wien
Favoritenstraße 9/E185
A-1040 Wien, Austria
{ finerer | salzer }@logic.at

Abstract

The Unified Modeling Language (UML) has become a universal tool for the formal object-oriented specification of hard- and software. In particular, UML class diagrams and so-called multiplicities, which restrict the number of links between objects, are essential when using UML for applications like the specification of admissible configurations of components.

In this paper we give a formal definition of the semantics of UML class diagrams and multiplicities. We extend results obtained in the context of Entity Relationship diagrams to cover UML specific extensions like the (non-)uniqueness attribute of binary associations. We show that the consistency of such specifications can be checked in polynomial time, and give an algorithm for computing minimal configurations (models). The core of our approach is a translation of UML class diagrams to Diophantine inequations.

1 Introduction

The *Unified Modeling Language* (UML) [15] has gained wide acceptance in software engineering as a universal formalism for object-oriented modeling, offering notations for describing class relationships, component systems, processes, use cases, and more. The *Object Constraint Language* (OCL) [14] can be used to impose additional constraints. In the past decade UML and OCL have been also considered for specifying configurations [12].

The term *configuration* as used in this paper refers to an arrangement of functional units according to their nature, number, and chief characteristics [13]. 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 crite-

ria 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 [20]. Compared to logic-oriented approaches, UML diagrams have the advantage that most (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 modeling. 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. [3,4] show how a wide range of UML/OCL elements can be translated to DL formulas. Other approaches use sub-classes of first-order logic (see e.g. [21]). 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 [4], called *ALUQT*, is EXPTIME-complete [2].

Calvanese and Lenzerini investigate is-a and cardinality ratio constraints in *Entity Relationship* (ER) schemata [5]. They solve the satisfiability problem with disequations and

so-called expansion mechanisms, and show that this task is decidable in EXPTIME. In [6] they discuss a framework for class-based representation formalisms like ER diagrams. They use the description logic $\mathcal{ALUN}\mathcal{I}$ 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 paper 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 use them to check the satisfiability (consistency) of dependency constraints in ER schemata [18]. Engel and Hartmann compute minimal solutions for semantic ER schemata with the Ford-Bellman algorithm [10]. 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 [7] and the *look-here approach* introduced with Merise [19] (see [16] for a discussion of the differences). Lenzerini and Nobili adhere to the latter view, while UML prescribes the former.

The paper is organised as follows. The next section describes those aspects of UML class diagrams that are relevant to this paper. Section 3 gives a rigorous definition of UML specifications as well as of configurations satisfying them. Section 4 presents a transformation of UML specifications to Diophantine inequations, and Section 5 describes algorithms for determining the solvability and for computing minimal solutions. Finally, Section 6 discusses open problems.

2 UML class diagrams

Class diagrams describe classes and associations in a static context. In general a class may have attributes and methods, but in this paper we are only interested in the relationships between objects of different classes. These are specified by so-called *associations*, which are tagged with *multiplicities* at each end. A multiplicity $m..n$ means that the number of partner objects has to be in the interval $[m, n]$. Moreover, each end of an association is marked with the property *unique* (“uniq”, the default) or *non-unique*



Figure 1. Example of UML class diagram

(“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. Additionally we impose lower bounds on the number of objects that instantiate a given class; these can be specified by simple OCL constraints or by some other means. For example, the diagram in Fig. 1 specifies that every object of class A needs to have 3 or 4 connections 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 .

3 Configurations and Specifications

To prove the correctness and completeness of our encoding 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 $[a, b] = \{i \mid a \leq i \leq b\}$, $[a, \infty] = \{i \mid i \geq a\}$, and $[]$ is synonymous for the empty set. Moreover, let Cl be a set of classes and R be a set of roles.

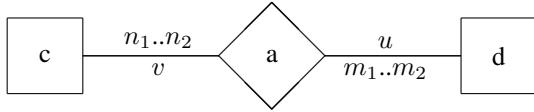
Definition 1 The tuple $\langle A, mult, uniq, lb \rangle$ is called *specification* if:

- $A \subseteq 2^{R \times Cl}$ 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.

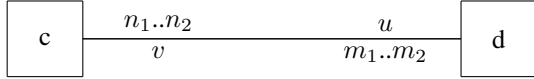
The specification is called *symmetric* if for every association $\{r_1 : c_1, \dots, r_n : c_n\}$ in A we have $uniq(r_1) = \dots = uniq(r_n)$.

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 $mult(r_2) = [m_1, m_2]$, $mult(r_1) = [n_1, n_2]$, $uniq(r_2) = u$ and $uniq(r_1) = v$ is depicted as



Binary associations are usually depicted as



Definition 2 The tuple $\langle O, L, class, ass, obj \rangle$ is called *configuration* if:

- 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 $l \in L$ satisfying $obj(l) = \{r_1 : o_1, \dots, r_n : o_n\}$.

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 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:

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

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 :

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

Definition 3 A config. $\langle O, L, class, ass, obj \rangle$ satisfies 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.

Example 4 Figure 2 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 As 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.

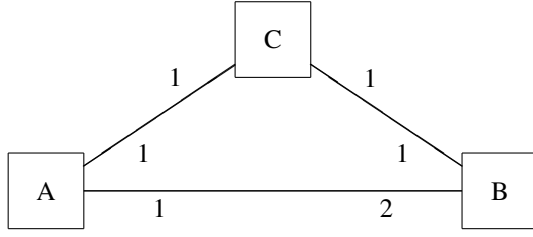


Figure 2. UML specification that is weakly but not strongly consistent

Example 5 Let S be the unsymmetric specification depicted in Fig. 1 with $lb(A) = lb(B) = 0$, and let $C = \langle O, L, class, obj \rangle$ be the configuration defined by $O = \{a, b\}$, $L = \{l_1, l_2, l_3\}$, $class(a) = A$, $class(b) = B$, and $obj(l_1) = obj(l_2) = obj(l_3) = \{r_1 : a, r_2 : b\}$. C satisfies S , since we have

$$\gamma_{r_1}(\{r_2 : b\}) = |\{a\}| = 1 \in [1, 2] = mult(r_1) \quad \text{and} \\ \delta(\{r_1 : a\}) = |\{l_1, l_2, l_3\}| = 3 \in [3, 4] = mult(r_2) .$$

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 for all links $l \in L$ such that $r : o \in obj(l)$ for some $o \in O$ the following conditions hold:

- $\delta(obj(l) \setminus \{r : o\}) \in mult(r)$, and
- if $uniq(r) = \text{uniq}$ then for all role-object tuples x, y and all links l_1, l_2 such that $obj(l_1) = \{r : o, x\}$ and $obj(l_2) = \{r : o, y\}$, $l_1 \neq l_2$ 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/nuniq` associated classes are connected by at most one link.

Example 7 This uniform view of `uniq/nuniq` does not work for unsymmetric specifications. Consider the specification and configuration of Example 5, where $uniq(r_1) = \text{uniq}$ but $uniq(r_2) = \text{nuniq}$. Let $r : o = r_2 : b$ and $x = y = r_1 : a$. Then we have $obj(l_1) = \{r : o, x\}$, $obj(l_2) = \{r : o, y\}$ 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.

4 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:

$$\varphi_{lb}(c) := x_c \geq lb(c)$$

$$\varphi_{mult}(r_1:c, r_2:d) := m_2 * x_c \geq n_1 * x_d$$

$$\varphi_{uniq_min}(r_1:c, r_2:d) := (x_d > 0 \implies x_c \geq n_1)$$

$$\varphi_{uniq_max}(r_1:c, r_2:d) := (m_1 > 0 \implies n_2 * x_d \geq x_c)$$

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`:

$$\psi_{nn}(x, y) := \varphi_{mult}(x, y) \wedge \varphi_{mult}(y, x)$$

$$\psi_{uu}(x, y) := \varphi_{mult}(x, y) \wedge \varphi_{uniq_min}(x, y) \wedge \varphi_{mult}(y, x) \wedge \varphi_{uniq_min}(y, x)$$

$$\psi_{un}(x, y) := \varphi_{mult}(x, y) \wedge \varphi_{uniq_min}(x, y) \wedge \varphi_{uniq_max}(x, y)$$

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 $\text{sc}(S)$, since we have $\psi_{nn}(r_1:c, r_2:d) = \psi_{nn}(r_2:d, r_1:c)$ and $\psi_{uu}(r_1:c, r_2:d) = \psi_{uu}(r_2:d, r_1:c)$.

Example 9 Let S be the specification $\langle A, \text{mult}, \text{uniq}, \text{lb} \rangle$ with $A = \{r_1 : c, r_2 : d\}$, $\text{mult}(r_2) = [3, 4]$, $\text{mult}(r_1) = [1, 2]$, $\text{lb}(c) = 1$, $\text{lb}(d) = 0$, and $\text{uniq}(r_1) = \text{uniq}(r_2) = \text{uniq}$. The sat-condition $\text{sc}(S)$ is the conjunction of

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

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 $[m_1, m_2] * x_c$ and $[n_1, n_2] * x_d$ overlap, i.e., they are satisfied if and only if there is a number k such that $m_1 * x_c \leq k \leq m_2 * x_c$ and $n_1 * x_d \leq k \leq n_2 * x_d$.

Remark 11 For each pair of classes c and d of an asymmetric association (w.l.o.g. assume $\text{uniq}(r_1) = \text{uniq}$), the sat-condition contains the constraints $m_2 * x_c \geq n_1 * x_d$ and $n_2 * x_d \geq x_c$, which express that the intervals $[1, m_2] * x_c$ and $[n_1, n_2] * x_d$ overlap, i.e., they are satisfied if and only if there is a number k such that $x_c \leq k \leq m_2 * x_c$ and $n_1 * x_d \leq k \leq n_2 * x_d$.

Theorem 12 A specification S is weakly consistent if and only if the formula $\text{sc}(S)$ is solvable.

Given a solution of $\text{sc}(S)$, Algorithm 1 completes the configuration by constructing admissible links. For the proof of the theorem and of the correctness of the algorithm see [11].

5 Solving Linear Diophantine Inequations

Several algorithms have been proposed for solving linear inequations over non-negative integers. Based on the work of [8] and [9], an algorithm without slack variables is presented in [1]. 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 [17] shows that the problem 2-IP (just two variables per inequation) is already NP-complete.

However, our class of inequations has a specific property: we need no upper bounds, i.e., no inequations of the form $c \geq ax + by$ or $c \geq ax$. We show in the following that

Algorithm 1 Constructing the links of a configuration

```

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

```

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 13 A UML-constraint is either an inequation of the form $b * y \geq a * x$, an inequation $y \geq a$, an implication of the form $a > 0 \implies b * y \geq x$ or an implication of the form $x > 0 \implies y \geq a$, where a, b are positive integers and x, y are variables. A UML-formula is a conjunction of UML-constraints.

Note that an inequation $a_1 > 0 \implies b * y \geq x$ can be transformed into an $b * y \geq a_2 * x$ (in this special case with $a_2 = 1$) inequation by simply testing whether a_1 is positive. This simplification can and should be done in a preprocessing step since the premise of the implication contains no variable.

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 .

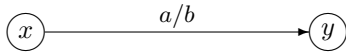
Proposition 14 *Let φ be a UML-formula, and let σ and τ be solutions of φ . Then $\min(\sigma, \tau)$, $\sigma + \tau$ and $a\sigma$ are also solutions of φ .*

Corollary 15 *The minimal solution of a UML-formula is unique.*

Corollary 16 *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 $b * y \geq a * x$:



Theorem 17 *Let I be a set inequations of the form $b * y \geq a * x$, 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 $O(|I| * |V|^2)$.*

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

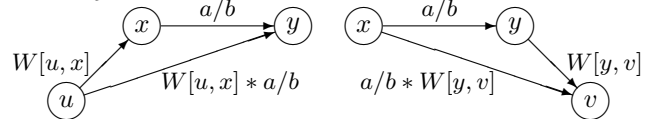
Algorithm 2 Compute paths with maximal weights

```

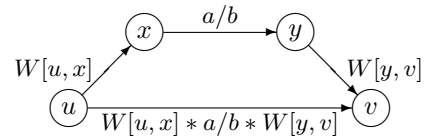
1: Let  $I$  be a set of input inequations  $b * y \geq a * x$ 
2: Let  $V$  be the set of variables occurring in  $I$ 
3: Let  $W$  be a  $|V| \times |V|$  matrix of non-negative rational
   numbers, initially set to zero.
4: Let  $P$  be a  $|V| \times |V|$  matrix of strings of variables,
   initially set to the empty strings.
5: for all inequations  $b * y \geq a * x$  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:     $w \leftarrow W[u, x] * a/b$ 
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:     $w \leftarrow a/b * W[y, v]$ 
20:    if  $w > W[x, v]$  and  $y \notin P[x, v]$  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:     $w \leftarrow W[u, x] * a/b * W[y, v]$ 
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  $(W, P)$ 

```

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 $O(|I| * (1 + |V| + |V| + |V| * |V|))$, i.e. $O(|I| * |V|^2)$. \square

Theorem 18 *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 $y \geq W[y, y] * y \geq W[x, y] * x$ 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 18. Note that the constraints φ_{lb} and φ_{uniq_min} do not affect strong consistency since they only fix lower bounds, which can always be satisfied by some configuration provided the corresponding variable is consistent.

Algorithm 3 Compute minimal rational solution

Require: Let I be a set of input inequations $a * x \leq b * y$, L the set of φ_{lb} conditions $n \leq x$ and U the set of φ_{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: **for** all variables y in V such that $x \neq y$ **do**
- 10: **if** $\sigma(x) * W[x, y] > \sigma(y)$ **then**
- 11: $\sigma(y) \leftarrow \sigma(x) * W[x, y]$
- 12: **end if**
- 13: **end for**
- 14: **for** all φ_{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

Theorem 19 *Given a set I of consistent input inequations $a * x \leq b * y$, φ_{lb} conditions $n \leq x$ and φ_{uniq_min} conditions $x > 0 \implies n \leq y$ 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 φ_{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 φ_{uniq_min} conditions in steps 14–19. Observe that φ_{uniq_min} conditions are triggered φ_{lb} conditions. Hence if a φ_{uniq_min} condition becomes active, we add it to the considered φ_{lb} conditions (step 17). \square

The time complexity of algorithm 3 is $O((|L| + |U|) * (|V| + |U|))$.

An integer solution for any variable v can be built by multiplying $\sigma(v)$ with $\text{lcm}\{b_x \mid x \text{ occurs in } I\}$ where b_x denotes the non-negative integer b in each inequation $a * x \leq b * y$. This is a valid operation by Theorem 20.

Theorem 20 *Let I be a system of inequations of the form $a * x \leq b * y$, 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 prove the two directions separately.

(\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 $n * \sigma(x)$ is an integer. Moreover, every multiple of a solution is again a solution, hence $n * \sigma$ is a non-negative integer solution of I . Since $\sigma(v) > 0$ we also have $n * \sigma(v) > 0$. \square

Theorem 21 *Given a set I of consistent input inequations $a * x \leq b * y$, φ_{lb} conditions $n \leq x$ and φ_{uniq_min} conditions $x > 0 \implies n \leq y$ algorithm 4 computes the minimal integer solution for all variables x, y in the graph representation of I .*

Proof: The argument 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 20 with $\text{lcm}\{b_x \mid x \text{ occurs in } I\}$ iterations, where b_x denotes the non-negative integer b in each inequation $a * x \leq b * y$. \square

Algorithm 4 Compute minimal integer solution

Require: Let I be a set of input inequations $a * x \leq b * y$, L the set of φ_{lb} conditions $n \leq x$ and U the set of φ_{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** $\lceil \sigma(x) * W[x, y] \rceil > \sigma(y)$ **then**
- 11: $\sigma(y) \leftarrow \lceil \sigma(x) * W[x, y] \rceil$
- 12: **end if**
- 13: **until** $\lceil \sigma(x) * W[x, y] \rceil = \sigma(y)$ for all variables x, y such that $x \neq y$
- 14: **for** all φ_{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

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

6 Conclusion

In this paper we described an efficient method for handling UML-specifications of configuration problems. We proposed to translate UML class diagrams to linear inequations over non-negative integers and to use these for checking consistency and for finding minimal solutions. We succeeded in giving a polynomial algorithm for checking consistency, and an algorithm for computing minimal solutions that performs better than other algorithms proposed in literature.

Our next step will be to extend the translation to inequations such that we can also handle n -ary associations (hyper-edges). In contrast to the look-here interpretation of ER diagrams, where n -ary associations can be treated in a similar way as binary ones, the look-across semantics of UML seem to require a new type of constraints. At the moment it is unclear whether n -ary associations are really es-

sential for specifying configurations. But their integration into our framework seems to be a worthwhile goal on its own. Another extension concerns OCL-constraints. OCL is much too powerful to be integrated to any reasonable extent, but certain constraints on the cardinality of classes might be mapped to inequations nevertheless.

We have not yet found a polynomial algorithm for constructing a minimal integer solution. Therefore we plan to improve the efficiency of our algorithm and either find a polynomial one or prove the hardness of the problem. In the latter case, suitable heuristics might improve the situation in practice. Another interesting issue is to find a finite description of *all* solutions, e.g. as linear combination of base vectors. This can be useful when adding a post-processing step that imposes further constraints on the desired solutions. One way to calculate these base vectors is to modify the algorithm by Ajili and Contejean [1]. However, an efficient algorithm for the quite specific form of our inequations has probably to be constructed along different lines.

So far we have only considered the problem of constructing a minimal configuration for a given specification. In practice, specifications are gradually extended over time. The corresponding configurations should evolve in a conservative manner. One usually does not want to reconstruct a whole train station just to integrate a new train switch. Therefore we will also consider the problem of extending configurations incrementally.

7 Acknowledgements

We would like to thank Andreas Falkner and Gottfried Schenner (Siemens Austria) for showing us real-world examples from the domain of railway interlocking systems and for presenting us the Siemens approach to configuration management. Moreover, we are grateful to Ana Paula Tomás (Universidade do Porto) for her advice on Diophantine inequations, in particular for hinting us at the work of J.Lagarias [17]. Finally, thanks to an anonymous referee of an earlier version of the paper who directed our attention to related work published in the context of ER diagrams.

References

- [1] F. Ajili and E. Contejean. Avoiding slack variables in the solving of linear diophantine equations and inequations. *Theoretical Computer Science*, 173(1):183–208, 1997.
- [2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [3] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on uml class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of*

Description Logics. CEUR Electronic Workshop Proceedings, <http://ceur-ws.org/Vol-44/>, 2001.

- [4] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1-2):70–118, 2005. <http://www.inf.unibz.it/~calvanese/papers-html/AIJ-2005.html>.
- [5] D. Calvanese and M. 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.
- [6] D. Calvanese, M. Lenzerini, and D. 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.
- [7] P. P.-S. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions Database Systems*, 1(1):9–36, 1976.
- [8] M. Clausen and A. Fortenbacher. Efficient solution of linear diophantine equations. *Journal of Symbolic Computation*, 8(1-2):201–216, 1989.
- [9] E. Contejean and H. Devie. An efficient incremental algorithm for solving systems of linear diophantine equations. *Information and Computation*, 113(1):143–172, 1994.
- [10] K. Engel and S. Hartmann. Constructing realizers of semantic entity relationship schemes. Technical report, Universität Rostock, Fachbereich Mathematik, 18051 Rostock, Germany, 1995.
- [11] I. Feinerer and G. Salzer. Consistency and minimality of uml class specifications with multiplicities and uniqueness constraints. Technical Report FS 2006/1, Technische Universität Wien, Vienna, Austria, 2006.
- [12] A. Felfernig, G. Friedrich, D. Jannach, and M. 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, 2002. Springer-Verlag.
- [13] T. A. for Telecommunications Industry Solutions. Atis telecom glossary 2000, 2000. Approved February 28, 2001 by the American National Standards Institute (ANSI). <http://www.atis.org/>.
- [14] O. M. Group. *Unified Modeling Language 2.0 Object Constraint Language Specification*, 2005. <http://www.omg.org/cgi-bin/doc?ptc/05-06-06> (visited 2007/01/28).
- [15] O. M. Group. *Unified Modeling Language 2.0 Superstructure Specification*, 2005. <http://www.omg.org/cgi-bin/doc?formal/05-07-04> (visited 2007/01/28).
- [16] S. Hartmann. Reasoning about participation constraints and chen's constraints. In K.-D. Schewe and X. 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.
- [17] J. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.

- [18] M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.
- [19] A. Rochfeld and H. Tardieu. MERISE: An information system design and development methodology. *Information & Management*, 6:143–159, 1983.
- [20] G. Schenner and A. 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.
- [21] G. Schenner and G. Fleischanderl. Modifying configurations with model finding. In *Proc. IJCAI 2003*, Acapulco, Mexico, 2003. Morgan Kaufmann.

Appendix: Proof of Theorem 12

Proof: We prove the two directions separately.

(\Rightarrow) Let $S = \langle A, mult, uniq, lb \rangle$ 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., $uniq(r_1) = uniq(r_2)$) 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 $\delta(\{r_1 : o\}) \in mult(r_2) = [m_1, m_2]$ and $\delta(\{r_2 : p\}) \in mult(r_1) = [n_1, n_2]$ for all objects o of type c and objects p of type d , we have

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

By remark 10, the existence of such a number k is expressed by the constraints φ_{mult} . Finally, if $uniq(r_2) = uniq$ (similar argumentation for $uniq(r_1)$) 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 :

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

For the asymmetric case (w.l.o.g. assume $uniq(r_1) = uniq$) let k denote the number of allowed links between all objects of class c and all objects of class d . Because of $\delta(\{r_1 : o\}) \in mult(r_2) = [m_1, m_2]$ and $\gamma_{r_1}(\{r_2 : p\}) \in mult(r_1) = [n_1, n_2]$ for all objects o of type c and objects p of type d , following conditions hold:

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

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 $C = \langle O, L, \text{class}, \text{obj} \rangle$ 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 11, 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 $\delta(\{r_c : o\}) = \lfloor k'/\sigma(c) \rfloor$ and $\delta(\{r_d : p\}) = \lfloor k'/\sigma(d) \rfloor$.

Note that if line 16 for the uniq–uniq case gets triggered $\delta(\{r_c : o\}) = \delta(\{r_c : o'\}) = k'/\sigma(c)$ for all objects o, o' of class c holds. 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 uniq–uniq case we obtain that for all objects o_1, o_2 of class c and all links l_1, l_2 such that $\text{obj}(l_1) = \{r_c : o, r_d : o_1\}$ and $\text{obj}(l_2) = \{r_c : o, r_d : o_2\}$, $l_1 \neq l_2$ 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 $\sigma(c) * m_1 \leq k \leq \sigma(c) * m_2$, i.e.,

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

Combining the two chains of inequations we obtain $m_1 \leq \delta(\{r_c : o\}) \leq m_2$, i.e. $\delta(\{r_c : o\}) \in \text{mult}(r_d)$. 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 $\lfloor k/\sigma(d) \rfloor \leq \delta(r_d : p) \leq \lceil k/\sigma(d) \rceil$. By the choice of k the inequation $n_1 \leq k/\sigma(d)$ holds. Combining both inequations we receive $n_1 \leq \delta(\{r_d : p\})$. 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 . Due to rigorous uniformity this concludes $n_1 \leq \gamma_{r_c}(\{r_d : p\})$, 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 over r_d roles (which in fact is $\delta(\{r_d : p\})$) are l_d/l_c , i.e., $k/\sigma(d)/k/\sigma(c) = \sigma(c)/\sigma(d)$. By constraint $\varphi_{\text{uniq_max}}$, i.e., $n_2 * x_d \geq x_c$, 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 nuniq end $\delta(\{r_c : o\}) \in \text{mult}(r_d)$ is derived as in the symmetric case. By dual arguments we derive $\gamma_{r_d}(\{r_c : o\}) \in \text{mult}(r_d)$ and $\delta(\{r_d : p\}) \in \text{mult}(r_c)$ for the mirrored nuniq–uniq case.

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