Journal of Computer Science Research
https://journals.bilpubgroup.com/index.php/jcsr

ARTICLE

On Software Application Database Constraint-driven Design and Development

Christian Mancas*[*], *Cristina Serban, Diana Christina Mancas

Math. & Computer Science Department, Ovidius University, Constanta, 900720, Romania

ABSTRACT

This paper presents a methodology driven by database constraints for designing and developing (database) software applications. Much needed and with excellent results, this paradigm guarantees the highest possible quality of the managed data. The proposed methodology is illustrated with an easy to understand, yet complex medium-sized genealogy software application driven by more than 200 database constraints, which fully meets such expectations.

Keywords: Database constraint-driven design and development; Database constraint; Data plausibility; Software architecture; Design and development; The (elementary) mathematical data model; MatBase

1. Introduction

Software design and development research started as a mathematical branch [1,2]. Since then and to the end of the previous millennia, the emphasis was put on tackling complexity and delivering high quality, derived from rigor, and user-friendliness software [3,4].

Despite good fundamental textbooks on software engineering [5-8], unfortunately, the state of the art in the field became largely dominated by technologies and glossy graphic user interfaces (GUI) [9], instead of principles and sound methodology. However, fortunately, there is still research and results inspired by the roots of this discipline. Among them, we were always interested in *constraint-driven approaches*.

1.1 Literature survey

Almost three decades ago, for example, Hoog et al. [3] put it forward as an alternative to the waterfall model. Then, Lano [10] added constraints to UML class diagrams and state machines in the framework of

*CORRESPONDING AUTHOR:
Christian Mancas, Math. & Computer Science Department, Ovidius University, Constanta, 900720, Romania; Email: christian.mancas@gmail.com

ARTICLE INFO
Received: 14 February 2023 | Revised: 28 February 2023 | Accepted: 1 March 2023 | Published Online: 10 March 2023
DOI: https://doi.org/10.30564/jcsr.v5i1.5476

CITATION
Mancas, C., Serban, C., Mancas, D.C., 2023. On Software Application Database Constraint-driven Design and Development. Journal of Computer Science Research. 5(1): 31-45. DOI: https://doi.org/10.30564/jcsr.v5i1.5476

COPYRIGHT
Copyright © 2023 by the author(s). Published by Bilingual Publishing Group. This is an open access article under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) License. (https://creativecommons.org/licenses/by-nc/4.0/).

model-driven development (MDD), advocating for constraint-driven development (CDD). In the aftermath, Demuth et al. [11] went further and explored constraint-driven modeling (CDM), extended by Rebmann et al. [12] who proposed to automate the generation of model constraints instead of generating entire models.

In parallel, constraint-driven approaches were considered as well in narrower subfields of software engineering. For example, Siddiqui [13] proposed his *Pike*, a tool for checking code conformance to specifications; Shrotri et al. [14] use it for Machine Learning; Ciortuz [15] applies it to concurrent parsing of a natural language.

Moreover, such approaches are used outside the software engineering realm as well. For example, in linguistics, Kumaran [16] extends correspondingly Noam Chomsky's *Agree*, while, in PCB hardware design, OrCAD [17] makes heavy use of the constraint-driven paradigm.

Getting back to software engineering, let us first note that most of the applications designed, developed, maintained, and used are database (db) ones: extremely few software applications of today are not managing databases (dbs). However, db constraints are not systematically considered in software engineering design and development approaches anymore. Moreover, what is very intriguing for us is the spreading of the JSON technology, which gives the false impression that there is no need for db design anymore: you just design objects for the applications and JSON is automatically mapping them into db tables, with all needed constraints.

We advocate a dual approach: you should carefully design and implement a db and then use an advanced tool of the 5th generation of programming languages, e.g., *MatBase* [18], to automatically generate accordingly the software application for managing that db. It is true that the Relational Data Model (RDM) [19,20], which is powering most of today's DB Management Systems (DBMS), as well as the NoSQL datastores are not at all suited for such an approach: RDM provides only six constraint types, while NoSQL, practically, only one of them. This is probably why even otherwise excellent recent textbooks on db software application design like, for example, the one by Kleppmann [21], is almost not even mentioning db constraints.

In fact, while software engineering is still craftsmanship, dbs are pure applied math, namely the naïve algebraic theory of sets, relations, and functions, plus the first-order predicate logic (FOPL). In particular, *db constraints* are formalized by closed FOPL expressions, while db queries by the open ones [20] (recall that a FOPL expression is closed whenever all of its variable occurrences are bound to at least one quantifier and open when at least one of them is free, i.e. not bound to any quantifier; for example, all variable occurrences within a SQL SELECT clause are free, while all those in either WHERE or HAVING ones are bounded to a universal quantifier).

MatBase is a prototype intelligent db and knowledge base management system, based mainly on the (Elementary) Mathematical Data Model ((E)MDM) [22], but also on the Entity-Relationship (E-R) one (E-RDM) [20,23], RDM, and Datalog [19,24]. Its (E)MDM GUI accepts mathematical db schemes, translates them into both RDM and E-RDM ones, and automatically generates corresponding db software applications for managing them.

(E)MDM provides 73 constraint types on sets, relations, and functions (that includes, either explicitly, or implicitly, the 6 relational ones provided by the RDM). All these 73 types belong to the Horn clauses class, the largest FOPL one for which the implication problem is decidable.

1.2 Paper outline

MatBase's strategy to enforce constraints (which was manually used by Mancas [9]), based on our proposed DB Constraint-Driven Design and Development (DBCDDD) approach, is presented in the next section of this paper.

The third section presents and discusses the results of applying it to an interesting sub-universe centered around the genealogy trees. The paper ends with conclusions and references.

2. The DB constraint-driven design and development approach in software engineering

2.1 Proposed methodology

The DB Constraint-Driven Design and Development (DBCDDD) approach that we are proposing in this paper is made up of the 6 methodological steps summarized in **Figure 1**.

2.2 Sub-universe analysis

You might want to apply in this step Algorithm *A*0 from Mancas [20], such as to obtain for the sub-universe of interest an E-R data model [20], which is made of the following 3 deliverables:

(i) A comprehensive set of E-R diagrams (E-RDs);

(ii) An associated set of restrictions (business rules);

(iii) An informal description of the corresponding sub-universe.

This E-R data model (the only one that business-oriented people may understand) should be obtained with the help of and, finally, negotiated with, and approved by our customers. The E-R GUI of *MatBase* [25] may be used to draw, store, and maintain E-RDs.

During this step, the domain-driven approach [5,7] is very useful as well.

Obviously, not even Artificial Intelligence (AI) might ever fulfill this task, but only, at most, help software architects!

2.3 Translation of the resulting E-R data model into a(n) (E)MDM scheme

This step, detailed in Algorithm *A*1 from Mancas [26], can be partially done automatically, with the help of an intelligent DBMS like *MatBase*, which is translating E-RDs into (E)MDM schemes, but only software architects may formalize restrictions (business rules) as FOPL constraints.

2.4 (E)MDM scheme validation and enhancement

For validation, you might want to apply in this step the Algorithm *A*2 from Mancas [26], to correct any modeling errors done in the first step (e.g., declaring a set as being of the relationship type when,

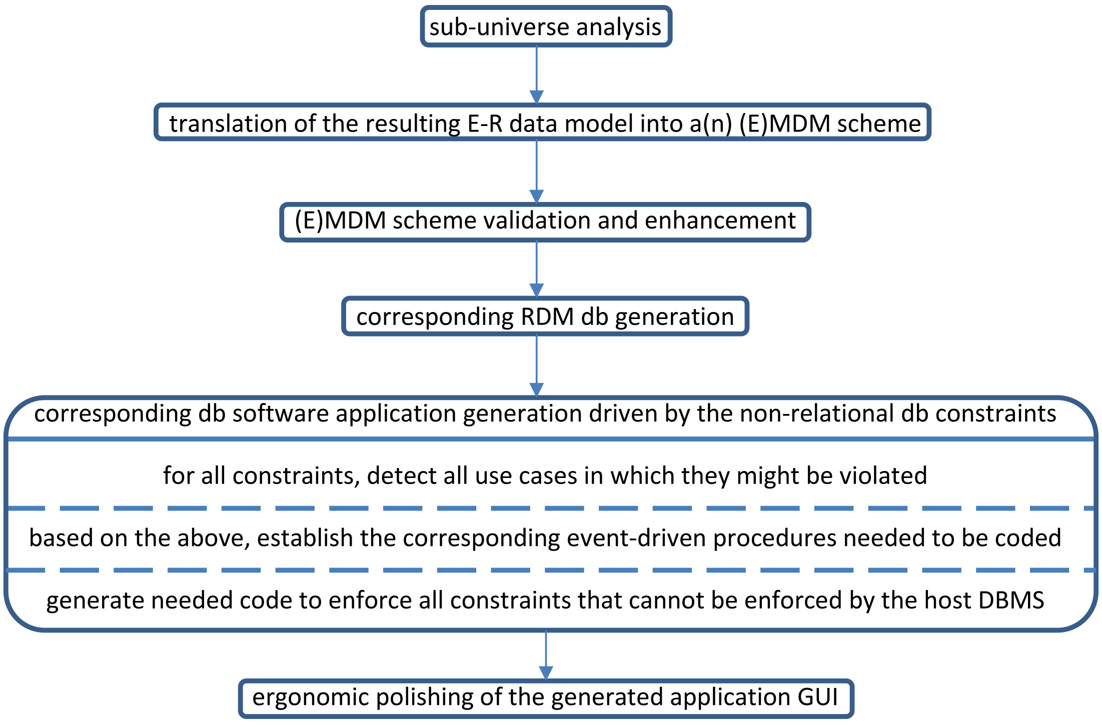
sub-universe analysis
translation of the resulting E-R data model into a(n) (E)MDM scheme
(E)MDM scheme validation and enhancement
corresponding RDM db generation
corresponding db software application generation driven by the non-relational db constraints
for all constraints, detect all use cases in which they might be violated
based on the above, establish the corresponding event-driven procedures needed to be coded
generate needed code to enforce all constraints that cannot be enforced by the host DBMS
ergonomic polishing of the generated application GUI

Figure 1. The DBCDDD methodology steps.

in fact, it can only be of the entity one or adding a constraint that does not exist in reality).

In our opinion, data correctness is utopian: for example, very probably, almost nobody knows or will ever know what the height of HM Queen Elizabeth II in Her last days on Earth was (moreover, we bet that most of us do not exactly know our current height, most of the time). Dually, anybody should be sure that the height in centimeters of any world person, any time, is a number between 20 (under which no premature baby managed to survive) and 275 (as the tallest man recorded was 272), so that only the values in this interval are plausible for the *Height* property of persons.

We understand by *plausible data value* (abbreviated as *plausible data*), in any sub-universe of discourse, any value of the data associated with a property (i.e., function codomain) that is satisfying all the business rules of that universe (or, equivalently, is not violating any of them). As such, *data plausibility*, i.e., the fact that a db instance stores only plausible values, is the highest possible form of data quality.

Beware that any existing constraint in the modeled sub-universe which is missing in your (E)MDM (or any other data model) scheme allows for storing unplausible data in your db (e.g., two persons with the same SSN (i.e., US Social Security Number), two countries with a same name, persons living a negative number of days or more than 120 years, etc.); dually, any constraint in your data model scheme that does not exist in the corresponding sub-universe prevents your software application end-users to store valid data in your db (e.g., enforcing for a *MARRIAGES* set/table the constraint *Husband* • *Wife* minimally one-to-one, i.e., declaring this set a relationship, instead of an entity type one, prevents storing data on remarriages, like, for example, the famous ones between Richard Burton and Elizabeth Taylor).

Moreover, enforcing redundant constraints (e.g., that *Mother*: *PEOPLE* $\rightarrow$ *PEOPLE* is not only acyclic, i.e., nobody may be his/her own mother, neither directly, nor indirectly, but also irreflexive and asymmetric, as acyclicity implies both of them), while not tampering with the db instances plausibility, is slowing down your corresponding software application for nothing. Consequently, redundant constraints should never be enforced, but only minimal constraint sets must be [22].

Dually, and much more important, we always need to make sure that our constraint sets are always coherent [22]: For example, if a constraint set contains both the constraint *CurrentCity acyclic*, i.e., no city may be its current one, neither directly, nor indirectly, and the constraint *CurrentCity reflexive*, i.e. the current city of any city is itself, then the corresponding *CurrentCity* column (of a *CITIES* table) would ever remain void (i.e., the corresponding function's image would always be the empty set), because acyclicity implies reflexivity, and any set containing both reflexivity and reflexivity is incoherent. Consequently, we should always remove incoherence from our constraint sets, preferably before coding an incoherent one.

Enhancements involve constraint discovery, as well as guaranteeing the coherence and minimality of the constraint sets. This second sub-step is the crucial one in the process and needs thorough deep thinking. Both (E)MDM and *MatBase* provide assistance algorithms for detecting all missing constraints [26-30], as well as for guaranteeing the coherence and minimality of the constraint sets [22,26].

Obviously, this step too may only be taken by software and db architects: For example, only humans may decide whether, in a given sub-universe, a function is a one-to-one, or a function product is minimally one-to-one or not (e.g. Mormons, some Arabs, some Chinese, etc. may have several simultaneous marriages, orthodox Christians may have at most 4 sequential marriages in a lifetime, catholic ones only one, except for exceptional papal approvals, etc.).

2.5 Corresponding RDM db generation

This step may be fully automated by an intelligent DBMS and *MatBase* is successfully doing it. Alternatively, you might do it manually, by using Algorithm *A*7 from Mancas [26].

This step also produces the sets of the non-relational constraints and of the relational ones that cannot be enforced by the target DBMS (e.g., MS SQL Server wrongly assumes implicitly that the NULLS set contains only one value, not infinite many ones; as such, it cannot enforce uniqueness constraints on table columns that might contain more than one null value). All constraints from both these sets must be enforced in the next step.

2.6 Corresponding db software application generation

This step is the core DBCDDD one: It takes as input the two above constraint sets that cannot be enforced by the DBMS host and generates the corresponding software application, which must enforce them instead. This step has the 3 sub-steps separated in **Figure 1** by dashed lines.

Especially this step might never be totally entrusted to anything or anybody else than a software and db architect. (E)MDM and *MatBase* are only assisting this process with Algorithm *A*9 from Mancas [26] and are automatically generating corresponding code whenever possible.

2.7 Ergonomic polishing of the generated application GUI

Even when using an intelligent tool like *MatBase*, at the end of the previous step you end up with only a set of MS Windows forms and their classes that are enforcing all the constraints. However, they must be ergonomically architectured in a hierarchy of forms and sub-forms that are called by a menu of the corresponding application.

Moreover, basic ergonomic principles should incite you to replace all context-independent (and, generally, incomprehensible to application's end-users, as they are hard to understand sometimes even by senior db developers) DBMS error messages with context-sensitive ones, to add facilities like pre-programmed queries and reports, navigation shortcuts between related data, to embellish the standard GUI with end-users fancied options, etc.

Obviously, all these may only be accomplished manually, by developers.

3. Results and discussion on applying DBCDDD to a genealogy sub-universe

Mancas [9] considered an extended genogram sub-universe, by adding to the genealogy trees data on countries, cities, monuments, marriages, and reigns of rulers over countries.

The MS SQL Server 2022 Developer edition was chosen as the application db host.

3.1 The sub-universe objects and their main properties

The corresponding E-R data model contains the following 13 object types (with their main properties in parentheses):

1) PERSONS (Name, Sex, Birth and PassedAway Dates and Cities, Mother, Father, Killer, BurialMonument, Family/Dynasty, Title, Nationality, Website, Picture, Notes);

2) DYNASTIES/FAMILIES (Name, Country, Founder, ParentHouse);

3) TITLES (Name);

4) MARRIAGES (Husband, Wife, Marriage, and Divorce Dates);

5) COUNTRIES (Name, Capital city, CurrentCountry, MainNationality);

6) CITIES (Name, Country, CurrentCity);

7) COUNTRIES_CAPITALS (Country, City, EstablishingYear);

8) CITIES_PICTURES (City, Picture, PictDescription);

9) MONUMENTS (Name, Type, City, Website, Notes);

10) MONUMENT_TYPES (Name);

11) MONUMENTS_PICTURES (Monument, Picture, PictDescription);

12) REIGNS (Person, Title, Country, Start and End Dates, Notes);

13) PARAMS (maxLifeYears, minMFertileAge, minFFertileAge, maxMFertileAge, maxFFertileAge, maxSurvivalMDays, maxSurvivalFDays).

The *Sex* property accepts 3 values: 'F' for females, 'M' for males, and 'N' for anything else (e.g., military occupations, international bodies administrations, etc.).

The corresponding structural E-RD [20] is shown in **Figure 2**.

3.2 The sub-universe constraints

This sub-universe is governed by 210 business rules. Their corresponding constraints are grouped as follows:

(i) 172 relational constraints, out of which:

- 21 domain (range) ones;
- 53 totality (not-null) ones;
- 2 default value ones;
- 12 primary key ones;
- 26 unique ones;
- 28 reference integrity (foreign key) ones;
- 30 tuple (check) ones.

(ii) 38 non-relational constraints.

Out of these 210 constraints, only the following 65 might raise issues (as the domain, totality, except for 2 of them, the ones for pictures, default, primary and foreign keys, as well as most of the tuple/check ones are simple to have them enforced by the MS SQL Server):

- C_1: There may not be two persons of the same dynasty (family) born in the same year and having the same names.
- C_2: No mother gives the same names to two of her children.
- C_3: No father gives the same names to two of his children.
- C_4: No person may live less than 0 days and more than *maxLifeYears* years.
- C_5: Mothers' sex must be 'F'.
- C_6: Wives' sex must be 'F'.
- C_7: Fathers' sex must be 'M'.
- C_8: Husbands' sex must be 'M'.
- C_9: Nobody may be his/her own mother, neither directly, nor indirectly (i.e., no ancestor, other than his/her mother, or descendant of somebody may be that somebody's mother).
- C_{10}: Nobody may be his/her own father, neither directly, nor indirectly (i.e., no ancestor, other than his/her father, or descendant of somebody may be that somebody's father).
- C_{11}: Nobody may be his/her ancestor or descendant.
- C_{12}: No woman may give birth before being *minFertileFAge* or after being *maxFertileFAge* years old, or after her death.
- C_{13}: No man may have a child before being *minMertileFAge* or after being *maxMertileFAge* years old, or more than *maxMSurvivalDays* after his death.
- C_{14}: Nobody may get married before being born or after death.

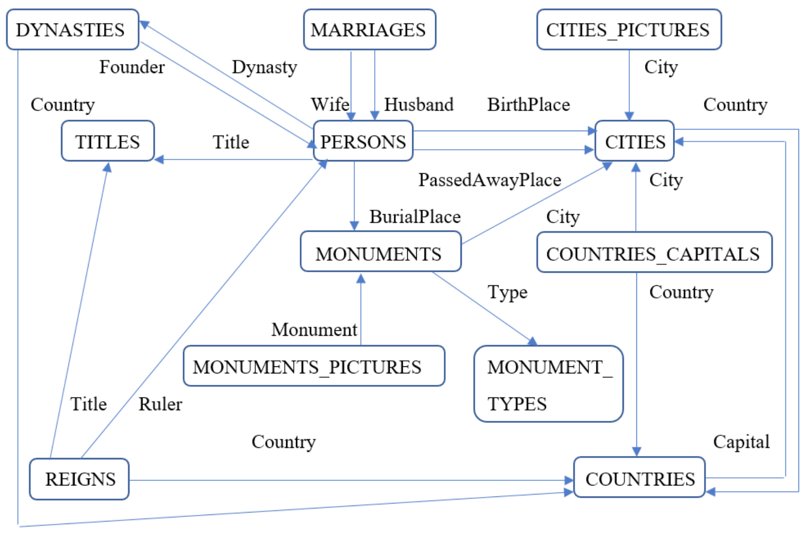

Figure 2. The structural E-RD of the genealogy db from Mancas [9].

- C_{15}: Nobody may divorce before being born or after death.
- C_{16}: Nobody may divorce before getting married.
- C_{17}: Nobody may get married twice on a same date.
- C_{18}: Nobody may get divorced twice on a same date.
- C_{19}: Nobody can get married while still being married.
- C_{20}: For any marriage, both spouses must be simultaneously alive for at least one day.
- C_{21}: No woman may be the wife of one of her ancestors or descendants.
- C_{22}: No man may be the husband of one of his ancestors or descendants.
- C_{23}: Nobody may be killed by somebody who was not alive when the assassination occurred.
- C_{24}: Nobody may belong to a dynasty (family) founded after his/her death.
- C_{25}: The founder of a dynasty (family) must belong to that dynasty or to its parent house.
- C_{26}: Nobody may have found more than one dynasty (family).
- C_{27}: There may not exist two dynasties with the same name.
- C_{28}: Any parent house must be established before any of its child dynasties.
- C_{29}: No dynasty (family) may be its ancestor or descendant, neither directly, nor indirectly.
- C_{30}: It does not make sense to store more than once a title.
- C_{31}: Nobody may reign before birth or after death.
- C_{32}: No country may be simultaneously ruled by two persons, except for spouses and for regencies.
- C_{33}: No reign may end before its start.
- C_{34}: It does not make sense to store more than once the fact that somebody started his/her rule in a country at any given date.
- C_{35}: It does not make sense to store more than once the fact that somebody ended his/her rule in a country at any given date.
- C_{36}: There may not be two countries having the same names.
- C_{37}: No country maybe its current one, neither directly, nor indirectly.
- C_{38}: No former country may be a current one.
- C_{39}: There may not be two cities of the same country having the same names.
- C_{40}: No city may be its current one, neither directly, nor indirectly.
- C_{41}: No former city may be a current one.
- C_{42}: The capital city of any country must either belong to that country, or to the current country of it, or to a former country whose current one is that country.
- C_{43}: No country establishes more than one city as its capital in any given year.
- C_{44}: It does not make sense to store more than once a picture from a city.
- C_{45}: Picture descriptions for the same city must be unique.
- C_{46}: It does not make sense to store more than once a picture of a monument.
- C_{47}: Picture descriptions for the same monument must be unique.
- C_{48}: There may not be two monuments in the same city having the same names.
- C_{49}: The website of a monument may not be shared by another monument.
- C_{50}: It does not make sense to store more than once a monument type.
- C_{51}: Whenever birth month and/or day are known, the birth year must be known too.
- C_{52}: Whenever the death month and/or day are known, the death year must be known too.
- C_{53}: Whenever the reign start month and/or day is known, the reign start year must be known too.
- C_{54}: Whenever the reign end month and/or day are known, the reign end year must be known too.
- C_{55}: Persons of sex 'N' may not have either parents or children, may not marry, and may not belong to dynasties (families).
- C_{56}: There may not be two persons having no

parents, no birth year, but the same name, sex, notes, and dynasty, or no dynasty.

- C_{57}: Nobody may have a brother as his/her father.
- C_{58}: Nobody may have a sister as his/her mother.
- C_{59}: City pictures are mandatory.
- C_{60}: Monument pictures are mandatory.
- C_{61}: There may not be more than one value for any application parameter.
- C_{62}: Parameter values may not be deleted.
- C_{63}: 0 < *minMFertileAge* < *maxMFertileAge* < *maxLifeYears*
- C_{64}: 0 < *minFFertileAge* < *maxFFertileAge* < *maxLifeYears*
- C_{65}: *maxSurvivalFDays* and *maxSurvivalMDays* parameter values may not be modified.

Out of these 65 constraints, 28 are relational ones, but only the following 9 out of them may be enforced by the MS SQL Server, namely: C_{27}, C_{30}, C_{36}, C_{39}, C_{43}, C_{48}, C_{50}, C_{63}, and C_{64}.

The 19 remaining ones (13 of type uniqueness, namely C_1, C_2, C_3, C_{17}, C_{18}, C_{26}, C_{34}, C_{35}, C_{44}, C_{45}, C_{46}, C_{47}, and C_{49}, as well as 4 of type tuple/check, namely C_4, C_{16}, C_{33}, C_{55}, and 2 of type totality, namely C_{59} and C_{60}) may not be enforced through the MS SQL Server, because the first 17 ones include at least one table column (which corresponds to a function defined on the set represented by its table, which corresponds in its turn to an object property) that accepts nulls, whereas the last two ones are on columns of type VARBINARY, on which no constraints are allowed. Consequently, all these 19 constraints must be enforced by the software application, just like the 38 non-relational ones.

Unfortunately, in the end, two of these 57 constraints may not be enforced at all, namely C_{44} and C_{46}, as large, good quality pictures (for both cities and monuments, in this case) may not be manipulated in memory either, not even by the Variant type of VBA (although they are linked or embedded as OLEDB objects).

3.3 The use cases that might violate the 55 constraints to be enforced through application code

Please note that, as expected, persons for whom passed away dates are null are considered still alive. Similarly, reigns for which end dates are null are considered still ongoing. Marriages for which divorce dates are nulls are considered still ongoing only while both spouses are alive.

Constraint C_1

(i) Current person's dynasty (family) is replaced by a not-null one;

(ii) Current person's name is modified;

(iii) Current person's birth year is replaced by a not-null one.

Constraint C_2

(i) Current person's mother is replaced by a not-null one;

(ii) Current person's name is modified when his/her mother is known.

Constraint C_3

(i) Current person's father is replaced by a not null one;

(ii) Current person's name is modified when his/her father is known.

Constraint C_4

(i) Current person's birth or/and passed away dates are replaced (for birth by a not null one);

(ii) For persons still alive, simply by the passing time (i.e., not when data is modified).

Constraint C_5

(i) Selecting as the mother of the current person somebody of sex 'M' or 'N';

(ii) Changing the current person's sex to 'M' or 'N' when that person is a mother.

Constraint C_6

(i) Selecting as the wife of current marriage somebody of sex 'M' or 'N';

(ii) Changing the sex of a wife to 'M' or 'N'.

Constraint C_7

(i) Selecting as the father of the current person somebody of sex 'F' or 'N';

(ii) Changing the current person's sex to 'F' or 'N' when that person is a father.

Constraint C_8

(i) Selecting as the husband of current marriage somebody of sex 'F' or 'N';

(ii) Changing the sex of a husband to 'F' or 'N'.

Constraint C_9

Might be violated only when, for the current person, is selected as his/her mother that person or a maternal ancestor or descendant of him/her.

Constraint C_{10}

Might be violated only when, for the current person, is selected as his/her father that person or a paternal ancestor or descendant of him/her.

Constraint C_{11}

(i) Selecting as the father of the current person somebody who is an ancestor or descendant of his/her mother;

(ii) Selecting as the mother of the current person somebody who is an ancestor or descendant of his/her father.

Constraint C_{12}

(i) Selecting as the mother of the current person somebody who does not satisfy this condition;

(ii) Modifying birth and/or death dates of a mother;

(iii) Modifying birth and/or death dates of a child of a known mother.

Constraint C_{13}

(i) Selecting as the father of the current person somebody who does not satisfy this condition;

(ii) Modifying birth and/or death dates of a father;

(iii) Modifying birth and/or death dates of a child of a known father.

Constraint C_{14}

(i) Selecting as a spouse of current marriage somebody who does not satisfy this condition;

(ii) Modifying marriage date;

(iii) Modifying birth and/or death dates of a spouse.

Constraints C_{15} and C_{16}

Let us consider constraint $C_{15'}$: Nobody may divorce before getting married or after death. Together with C_{14}, $C_{15'}$ obviously imply both C_{15} and C_{16}; consequently, we replace them with $C_{15'}$, which might be violated only in the following 3 use cases:

(i) Selecting as a spouse of current marriage somebody who does not satisfy this condition;

(ii) Modifying marriage and/or divorce dates for the current marriage;

(iii) Modifying the death date of a spouse.

Constraint C_{17}

(i) Replacing the marriage date for the current marriage with a not-null one;

(ii) Modifying a spouse of the current marriage.

Constraint C_{18}

(i) Replacing the divorce date for the current marriage with a not-null one;

(ii) Modifying a spouse of the current marriage.

Constraint C_{19}

(i) Selecting as a spouse of current marriage somebody who does not satisfy this condition;

(ii) Modifying marriage and/or divorce dates for the current marriage.

Constraint C_{20}

(i) Selecting as a spouse of current marriage somebody who does not satisfy this condition;

(ii) Modifying marriage and/or divorce dates for the current marriage;

(iii) Modifying birth and/or death dates of a spouse.

Constraint C_{21}

Might be violated only when, for the current marriage, is selected as husband somebody who is an ancestor or descendant (either maternally or/and paternally) of the corresponding wife.

Constraint C_{22}

Might be violated only when, for the current marriage, is selected as wife somebody who is an ancestor or descendant (either maternally or/and paternally) of the corresponding husband.

Constraint C_{23}

(i) Selecting as a killer of the current person somebody else who does not satisfy this condition;

(ii) Modifying birth and/or death dates of a killer or/and the death date of the current person.

Constraint C_{24}

(i) Selecting as the founder of the current dynasty somebody who does not satisfy this condition;

(ii) Modifying the birth date of a founder of a dynasty;

(iii) Modifying birth and/or death dates of a member of a dynasty;

(iv) Replacing the current person's dynasty with a not-null one.

Constraint C_{25}

(i) Selecting as a founder of the current dynasty a known person;

(ii) Modifying the dynasty of its founder;

(iii) Replacing the current dynasty's parent house when the current dynasty's founder is not null.

Constraint C_{26}

C_{26} is redundant, as implied by C_{25}: Any founder belonging to its dynasty may not belong to another one as well.

Constraint C_{28}

(i) Selecting as founder of the current dynasty a known person;

(ii) Modifying the birth date of the dynasty founder;

(iii) Replacing the current dynasty's parent house with a not-null one.

Constraint C_{29}

Might be violated only when, for the current dynasty, is selected as the parent house either the current dynasty or one of its ancestors or descendants.

Constraint C_{31}

(i) Selecting as ruler of current reign somebody who does not satisfy this condition;

(ii) Modifying birth and/or death dates for a ruler;

(iii) Modifying start and/or end dates of the current reign.

Constraint C_{32}

(i) Selecting as co-ruler of a reign somebody who does not satisfy this condition;

(ii) Modifying birth and/or death dates for a co-ruler;

(iii) Modifying marriage and/or divorce dates for a co-ruler;

(iv) Modifying start and/or end dates of the current reign;

(v) Modifying the country of the current reign;

(vi) Modifying the title of a co-ruler.

Constraint C_{33}

Might be violated only when, for the current reign, start and/or end dates are modified.

Constraint C_{34}

(i) Modifying the start date of the current reign;

(ii) Modifying the country of the current reign;

(iii) Modifying the ruler of the current reign.

Constraint C_{35}

(i) Modifying the end date of the current reign;

(ii) Modifying the country of the current reign;

(iii) Modifying the ruler of the current reign.

Constraint C_{37}

Might be violated only when, for a country, is selected as its current one itself or one of its former ones.

Constraint C_{38}

Might be violated only when, for a country, is selected as its current country or a former one.

Constraint C_{40}

Might be violated only when, for a city, is selected as its current one itself or one of its former ones.

Constraint C_{41}

Might be violated only when, for a city, is selected as its current city or one of its former ones.

Constraint C_{42}

(i) Selecting for the current country in COUNTRIES_CAPITALS a city that does not satisfy this condition;

(ii) Modifying for a country occurring in COUNTRIES_CAPITALS its current one;

(iii) Modifying for a capital occurring in COUNTRIES_CAPITALS its current city;

Constraint C_{45}

(i) Replacing the description of the current city picture with a not-null one;

(ii) Replacing the city of the current city picture with another one.

Constraint C_{47}

(i) Replacing the description of the current monument picture with a not-null one;

(ii) Replacing the monument of the current monument picture with another one.

Constraint C_{49}

Might be violated only when replacing the website URL of a monument with a not null one.

Constraint C_{51}

Might be violated only when modifying the birthday and/or month and/or year of a person.

Constraint C_{52}

Might be violated only when modifying the death day and/or month and/or year of a person.

Constraint C_{53}

Might be violated only when modifying the start day and/or month and/or year of a reign.

Constraint C_{54}

Might be violated only when modifying the end day and/or month and/or year of a reign.

Constraint C_{55}

(i) Selecting a not null dynasty, father, or mother for a person of sex 'N';

(ii) Replacing the sex value of a person with 'N'.

Constraint C_{56}

(i) Attempting to enter corresponding duplicate data for a new person;

(ii) Replacing the mother and/or father and/or birth year of the current person with nulls;

(iii) Replacing name or/and sex or/and notes or/and dynasty of the current person.

Constraint C_{57}

(i) Adding/replacing a brother to the current person;

(ii) Adding/replacing the father of the current person.

Constraint C_{58}

(i) Adding/replacing a sister to the current person;

(ii) Adding/replacing the mother of the current person.

Constraint C_{59}

Might be violated only when adding to the current city a picture description without a picture.

Constraint C_{60}

Might be violated only when adding to the current monument a picture description without a picture.

Constraint C_{61}

Might be violated only when a second line is saved in the PARAMETERS table.

Constraint C_{62}

(i) Replacing a parameter value with a null one;

(ii) Deleting the only line of the PARAMETERS table.

Constraint C_{63}

Might be violated only when modifying the values of at least one of these 3 parameters.

Constraint C_{64}

Might be violated only when modifying the values of at least one of these 3 parameters.

Constraint C_{65}

Might be violated only when modifying the value of at least one of these 2 parameters.

3.4 Establishing the corresponding event-driven procedures needed to be coded

This sub-step heavily depends on the platform used for coding the application. For example, *MatBase*, which has two versions: -one for students and small dbs and a professional one- that uses VBA and C#, respectively. Mancas [9] opted for VBA, which is both simpler, very robust, and provides an extensive set of data-oriented events and associated event-driven procedures.

It is out of the scope of this paper to enter into details on software application development on any platform, as this would need at least another 20 pages per platform and would not be of any academic, but only of technological interest.

The only general aspect about this sub-step is the fact that there are two possible algorithmic approaches to enforce db constraints in software applications, just like in healthcare, namely:

(i) *preventively* (i.e., providing users to choose from only plausible data in combo-boxes),

(ii) *curatively* (i.e., letting users enter desired data and reject unplausible ones).

The preventive ones are the best and, for example, in VBA they may be coded in the *Form_Current* event-driven procedures, which are automatically called each time the cursor is set on another data line of the current form. For example, in the *DYNASTIES* form, this procedure should dynamically modify the SQL SELECT statements that compute the combo-boxes *Founder* and *ParentHouse* and then re-query them, such as to eliminate from *ParentHouse* the current dynasty and from *Founder* all persons that are not belonging to either the current dynasty or its parent house, as well as those dead before the birth of the current founder (thus preventively enforcing constraints C_{24}, C_{25} , and C_{29}, respectively).

Sometimes, however, this is not possible (not even for all combo-boxes and all types of constraints involving their corresponding columns, hence functions). For example, to enforce constraint C_{49} you can only let the user type any desired text string in the *Website* text-box control of the *MONUMENTS* form's current data line and then reject it within the *Website_BeforeUpdate* VBA event-driven procedure (corresponding to the *Validating* event type of .NET) if that URL is already stored in the db for another monument.

3.5 Comparative analysis

In Mancas [9], state-of-the-art analysis of genealogy software applications available on the market was conducted as well, starting from the No1Reviews.com website post on the top 10 of such applications in 2022 [31]. Only 8 of them have been analyzed (as one is only for Apple hardware and software and the other is a website builder not freely available for evaluation) and only 3 of them provide a rudiment of data quality consideration: For a few unplausible values (e.g. passed away date less than birth one) they warn you and ask a confirmation message to which, unfortunately, you can answer *Yes*, thus saving that unplausible data in their dbs. In all 8 of them we easily manage to save aberrantly unplausible data, like persons living centuries, getting married or/and baptized before birth or after death, mothers of sex 'M', fathers of sex 'F', persons being buried before death, etc.

Unfortunately, this is not an exception: Such software applications abound in all fields, not only in the genealogy one. Some might say that the corresponding software companies lack software and/or db architects or that all fine ones are working only for giants like Microsoft, Google, Apple, Tesla, etc.

We strongly believe, however, that the main reason for this catastrophic reality is that, on one hand, software engineering treats db applications just as the not-db ones and, on the other, it completely lacks consideration of the main asset of any db application, namely its managed data quality. And, as we've explained, data quality may be guaranteed only by plausible data values and data plausibility may be guaranteed only by discovering all business rules governing the considered sub-universes and enforcing all their corresponding constraints.

This is why we consider that our proposed db constraint-driven design and development methodology described in this paper is a crucial approach to take towards the delivery of high-quality software db applications, not only exhibiting glossy GUIs, but, especially, guaranteeing the highest quality possible of the managed data.

Using the DB Constraint-Driven Design and Development approach, the genogram software application described in Mancas [9] and in the previous section of this paper successfully and elegantly enforced all 208 constraints governing this sub-universe that can be enforced with the currently available technologies. The contrast between this application and the ones considered in reviews [31] as the best ones in this field could not be more spectacular.

4. Conclusions and further work

We introduced a novel database constraint-driven methodology for designing and developing software database applications. We exemplified it with a complex medium-sized software database application for managing genograms. We argued that, using this methodology, this application guarantees the highest possible quality of the data it is managing, whereas most of the similar applications available and considered to be the best ones in this field have almost no concern at all about data quality.

Moreover, although Mancas [9] used this paradigm manually, our previous research and the *MatBase* prototype embedding it provide powerful tools to program while modeling, which is the future of software, as fewer and fewer developers and testers, while more and more architects and designers will soon be needed with the generalization of automatic code generation.

Further work is needed to automate software applications' code generation for the (E)MDM general object constraints [22,26] in *MatBase*.

Author Contributions

Dr. Christian Mancas is the author of the software application DB Constraint-Driven Design and Development approach, which he was teaching for decades to his MSc. students with both the Math. & Computer Science Dept. of the Ovidius University at Constanta, Romania, and the English Stream Computer & Telecomm. Engineering Taught in Foreign Languages Dept. of the Politehnica University at Bucharest, Romania. Dr. Mancas wrote the first 2 sections of this paper, plus its last sentence (the one on further work above). Miss Diana Christina Mancas wrote the rest of it. Associate Professor Cristina Serban is the scientific coordinator of her work, as well as for her entire MSc. Dissertation Thesis [9].

Conflict of Interest

There is no conflict of interest.

Funding

This research received no external funding.

Acknowledgement

We are grateful to Mihaela Virginia Mancas for her thorough revision of the final manuscript, which, hopefully, thus got rid of any typo or syntax errors and is fully intelligible, as well as to the whole JCSR editorial team for their kind and competent assistance.

References

[1] Daylight, E.G., Niklaus, W., Hoare, T., et al., 2012. The dawn of software engineering: From Turing to Dijkstra. Lonely Scholar bvba: Belgium.

[2] Dijkstra, E.W., 1982. Selected writings on computing: A personal perspective. Springer Verlag: NY, Heidelberg, Berlin.

[3] Hoog, R. de, Jong, T. de, Vries, F. de, 1995. Constraint-driven software design: An escape from the waterfall model. 7(3), 48-63. DOI: https://doi.org/10.1111/j.1937-8327.1994.tb00637.x

[4] Hunt, A., Thomas, D., 1999. The pragmatic programmer: From journeyman to master. Addison-Wesley Professional: IL, USA.

[5] Evans, E., 2003. Domain-driven design: Tack-

ling complexity in the heart of software. Addison-Wesley Professional: IL, USA.

[6] Taylor, R.N., Medvidovic, N., Dashofy, E.M., 2010. Software architecture: Foundations, theory, and practice. Wiley: NJ, USA.

[7] Vernon, V., 2016. Domain-driven design distilled. Addison-Wesley: IL, USA.

[8] Ousterhout, J., 2021. A Philosophy of Software Design, 2nd edition. Yaknyam Press: CA, USA.

[9] Mancas, D. C., 2023. Design and development of a DB software application for managing genealogical trees [Master's thesis]. Constanta, Romania: Ovidius University. p. 670.

[10] Lano, K., 2008. Constraint-driven development. Information and Software Technology. 50(5), 406-423.
DOI: https://doi.org/10.1016/j.infsof.2007.04.003

[11] Demuth, A., Lopez-Herrejon, R.E., Egyed, A. (2012). Constraint-Driven Modeling through Transformation. In: Hu, Z., de Lara, J. (editors), Theory and Practice of Model Transformations. ICMT 2012. Lecture Notes in Computer Science. 7307, 248-263.
DOI: https://doi.org/10.1007/978-3-642-30476-7_17

[12] Rebmann, A., Weidlich, M., Aa, H. van der (editors), 2022. GECCO: Constraint-driven abstraction of low-level event logs. 38th IEEE International Conference on Data Engineering; 2022 May 9-12; Kuala Lumpur, Malaysia. USA: IEEE. pp. 150-163.
DOI: https://doi.org/10.1109/ICDE53745.2022.00016

[13] Siddiqui, J.H., 2012. Improving Systematic Constraint-driven Analysis using Incremental and Parallel Techniques [PhD thesis]. USA: University of Texas at Austin. [cited 2023 Feb 14]. Available from: https://repositories.lib.utexas.edu/bitstream/handle/2152/19568/siddiqui_dissertation_201221.pdf?sequence=1&isAllowed=y

[14] Shrotri, A.A., Narodytska, N., Ignatiev, A., et al., 2022. Constraint-driven explanations for black box ML models. Proceedings of the AAAI Conference on Artificial Intelligence. 36(8), 8304-8314.
DOI: https://doi.org/10.1609/aaai.v36i8.20805

[15] Ciortuz, L., 1997. Constraint-Driven Concurrent Parsing Applied to Romanian Transitive VP [Internet]. Proceedings of the International Workshop on Parsing Technologies [cited 2023 Feb 14]. Available from: https://aclanthology.org/1997.iwpt-1.26.pdf

[16] Kumaran, E., 2022. Constraint-driven Agree. Proceeding of Linguist Society America. 7(1), 5282.
DOI: https://doi.org/10.3765/plsa.v7i1.5282

[17] OrCAD, 2023. Integrated Front-To-Back Constraints for Right First Time Designs [Internet]. [cited 2023 Feb 14]. Available from: https://www.orcad.com/tech-solutions/constraint-driven-design

[18] Mancas, C., 2019. *MatBase*—A tool for transparent programming while modeling data at conceptual levels. Computer Science & Information Technology (CSITEC 2019). AIRCC Pub. Corp.: Chennai, India. pp. 15-27.
DOI: https://doi.org/10.5121/csit.2019.91102

[19] Abiteboul, S., Hull, R., Vianu, V., 1995. Foundations of databases. Addison-Wesley: IL, USA.

[20] Mancas, C., 2015. Conceptual data modeling and database design: A completely algorithmic approach. Volume I: The shortest advisable path. Apple Academic Press/CRC Press (Taylor & Francis Group): FL, USA.

[21] Kleppmann, M., 2016. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly: UK.

[22] Mancas, C., 2018. *MatBase* constraint sets coherence and minimality enforcement algorithms. Advances in databases and information systems. Springer: Switzerland. pp. 263-277.
DOI: https://doi.org/10.1007/978-3-319-98398-1_18

[23] Thalheim, B., 2000. Entity-relationship modeling: Foundations of database technology. Springer Berlin: Heidelberg.

[24] Mancas, C., Dragomir, S., 2004. *Matbase* Datalog Subsystem Metacatalog Conceptual Design [Internet]. Proceedings of the IASTED Conference on Software Engineering and Applications,

November 9-11, 2004, MIT, Cambridge, MA, USA. Acta Press: Canada. pp. 34-41 [cited 2023 Feb 14]. Available from: https://www.actapress.com/PaperInfo.aspx?PaperID=19050&reason=500

[25] Mancas, C., Mancas, S., 2005. *Matbase* Entity-Relationship Diagrams Subsystem Metacatalog Conceptual Design [Internet]. IASTED International Conference on Databases and Applications, Part of the 23rd Multi-Conference on Applied Informatics, Innsbruck, Austria. pp. 83-89. Acta Press: Canada. [cited 2023 Feb 14]. Available from: https://www.actapress.com/PaperInfo.aspx?PaperID=19050&reason=500

[26] Mancas, C., 2023. Conceptual data modeling and database design: A completely algorithmic approach. Volume II: Refinements for an Expert Path. Apple Academic Press/CRC Press (Taylor & Francis Group): FL, USA.

[27] Mancas, C., 2016. Algorithms for key discovery assistance. BIR 2016, lecture notes in business information processing. Springer: Switzerland. pp. 261, 322-338.
DOI: https://doi.org/10.1007/978-3-319-45321-7_23

[28] Mancas, C., 2019. *MatBase* E-RD cycles associated non-relational constraints discovery assistance algorithm. Intelligent computing. Springer: Switzerland. pp. 390-409.
DOI: https://doi.org/10.1007/978-3-030-22871-2_27

[29] Mancas, C., 2019. *MatBase* autofunction non-relational constraints enforcement algorithms. IJCSIT. 11(5), 63-76.
DOI: https://doi.org/10.5121/ijcsit.2019.11505

[30] Mancas, C., 2020. On detecting and enforcing the non-relational constraints associated to dyadic relations in *MatBase.* Journal of Electronic & Information Systems. 2(2), 1-8.
DOI: https://doi.org/ 10.30564/jeisr.v2i2.2090

[31] No1Reviews.com [Internet]. Reviews of the Top 10 Genealogy Software of 2023 [cited 2023 Feb 14]. Available from: https://genealogy-software.no1reviews.com