https://doi.org/10.1590/jatm.v14.1278 THEMATIC SECTION | AVIATION SAFETY AND CONTINUED AIRWORTHINESS

A Customized Data Model for an Integrated Process of Requirements and Configuration Management

Sérgio Ricardo de Freitas Oliveira[1,*], Guilherme Conceição Rocha[2], Donizeti de Andrade[2]

1.Lanlink Informática – Fortaleza/CE, Brazil. 2.Departamento de Ciência e Tecnologia Aeroespacial – Instituto Tecnológico de Aeronáutica – Divisão de Engenharia Aeronáutica – São José dos Campos/SP, Brazil.

*Correspondence author: sergio.r.f.oliveira@ieee.org

ABSTRACT

This work aims to propose a customized, open-source data model for supporting an integrated process of Requirements and Configuration Management that allows for keeping track of which requirement originated each part, subsystem and system—and vice versa—, over their entire life cycle. That model might prospectively contribute to the processes of Aeronautical Engineering design and Continued Airworthiness in organizations that do not want either to invest in complex existing commercial, proprietary software packages or to train their professionals for that purpose. The complete source code of the model is provided in Python and SQL languages.

Keywords: Requirements; Engineering; Configuration; Management; Safety.

INTRODUCTION

Design methodologies have been developed since the 1950s. According to Jänsch and Birkhofer (2006), the first guideline in this field was developed in 1954 by Fritz Kesselring—a Swiss electrical engineer—who, in his book *Technische Kompositionslehre* (Theory of Technical Composition) described a *Wegleitung zur Erfindung* (Guidance for inventions). Kesselring himself, along with Friedrich Hansen—a German mechanical engineer—, gave rise to the *Verein Deutscher Ingenieure* (VDI) directive 2222 to establish a "Project Methodology," in 1973. The VDI 2222 guideline has four phases: clarification of the task, conceptual design, embodiment design and detailed design, and already foresaw adaptations of requirements from one phase to another (Jänsch and Birkhofer 2006). VDI 2222 was replaced in 1993 by the VDI 2221 directive – "Systematic Approach to the Development and Design of Technical Systems and Products" which is still available (VDI 2017) and is based on VDI 2222 and on the principles of Systems Engineering (Jänsch and Birkhofer 2006).

According to Kenneth J. Schlager (1956), the term "Systems Engineering" was probably first used by Bell Telephone Laboratories. Buede (2009) reports that it appeared supposedly in the 1940s, and "had its development leveraged in World War II" (Buede 2009, p. 6-7; INCOSE 2017). Therefore, the recognition of the importance of requirements definition, allocation and management comes

Received: May 31, 2022 | **Accepted:** Sept 08, 2022
Peer Review History: Single-Blind Peer Review.
Section editor: Joana Ribeiro

from a long time. So much so, according to Buede (2009), the central problem of a project in Systems Engineering is defined in terms of requirements.

Over the years, modeling languages have been developed to try to give more formalism to requirements elicitation and management. One of the most notorious, if not the most notorious, is SysML, which is an open-source language proposed by the Object Management Group (OMG) and derived from the Unified Modelling Language (UML), that uses standardized constructs for analyzing, designing, specifying, verifying, and validating complex system (Ramos *et al.* 2012). SysML and UML offer a mix of data models, functional models and database models.

Time also brought sophistications to Systems Engineering, one of them being the Model-Based Systems Engineering (MBSE), which emphasizes the application of rigorous, formal analysis methods—usually computational models—throughout the design and development process (Blanchard and Blyer 2016, p. 27). Such a greater rigor is favored by substitution of paper documents for electronic ones. Here, the word "documents" may refer to requirements specifications, engineering diagrams, risk matrices, configuration controls, change controls, among many others.

Because MBSE demands formalism in requirements specification, SysML seems to be the ideal method for that objective, since it includes specific constructs for this purpose. However, it turns out that, in practice, the use of the SysML language is still limited, perhaps because learning it takes a lot of time and effort, or their diagrams may seem counterintuitive and difficult to understand by those unfamiliar with the language. To make things worse, the scenario is that MBSE be adopted when many multidisciplinary groups of people are working together on the same project. So, it is possible that some or many of those persons may not be versed in SysML, which makes it difficult for them to understand the diagrams and the logic of the language.

Bone and Cloutier (2010) report that, in a survey conducted by the OMG in 2009, one of the main sponsors of the SysML initiative regarding how SysML and MBSE had been used, 72.9% of the respondents agreed that "the primary purpose of the [MBSE] model is to improve the quality of requirements and design to reduce downstream defects" (p. 3). However, in that same survey, one of the key findings was that "the idea that MBSE and SysML require a steep learning curve is a theme that is found throughout the survey when asked open-ended questions" (p. 3). It is also interesting to note that half of the respondents agreed that "there should be an update to SysML in the next three years" (p. 4) but only in May 2017 OMG issued a new version of its SysML language (OMG 2017).

There is a very comprehensive MBSE open-source tool called Capella (Thales 2018), but it is much more than a requirement management software tool, and it is not web-based.

There are commercial alternatives from traditional vendors for Requirements Management that do not necessarily rely on SysML. Typically, they are software packages that implement some form of framework for Systems Engineering or for MBSE. To name some: IBM's DOORS and Dassault's 3DEXPERIENCE. Unfortunately, they are expensive—thousands of dollars per year (Software Engineering Research Group 2012)—, and/or do not support web-based usage. For some corporations, the cost for such tools might not even be the main problem when compared to other costs of their typical programs: it might be, rather, the need of one more hard-to-learn, complicated tool to deploy and disseminate. Bonnet *et al.* (2015, p. 3) reported that:

> [...] for several reasons, the large-scale deployment of solutions based on UML […] did not succeed [in their organization—the Thales Group—, where] […] software architects felt comfortable with UML, [but] systems and hardware architects did not – and SysML was not considered as fully bridging the gap.

It is plausible that an open-source, web-based, cloud-enabled software tool specific for RE and CM might potentially have a lower cost of ownership, be less *complicated* to deploy and easier to learn.

And there always exists the possibility of using spreadsheets—e.g., Excel—or text editors—e.g., Word—for basic requirements management. George Koelsch (2016), a systems engineer with more than 40 years of experience in writing requirements for the U.S. Army and U.S. Federal Government, in his book *Requirements Writing for System Engineering*, compared seven tools for requirements writing—varying from a simple typewriter to sophisticated software like CaliberRM, ReqPro, ALMComplete and DOORS. His comparison considered 15 characteristics: Capture all the requirements; Update the baseline quickly; Capture additional data; Visually represent all information; Support development methodologies; Easily search various fields; Generate a hard copy; Span multiple projects; Not too complex; Not limit the engineer; Good documentation; Flexible; Ease of installation; and Good online help and Cost. He concluded that:

> [...] with cost included, Excel is second only to the top application in the industry, DOORS, by a slight margin. If you remove cost, Excel is still second to DOORS but very comparable to all four of the dedicated requirements tools examined. (Koelsch 2016, p. 262)

Unfortunately, those who have already used electronic spreadsheets for writing requirements probably felt many deficiencies in referential integrity—e.g., keeping track of which requirement depends on which requirement—, keeping track of changes, phases, authoring privileges, etc., to name only a few difficulties.

In the light of what is stated above, the objective of this work is to define an open-source, customized data model that can be used by a software application for an integrated Requirements and Configuration Management process.

METHODS

It is plausible, given the public datasheets of existing commercial requirements management software, that most of them use some form of Relational Database Management System internally; so, it is not an innovation to propose a new approach to do that. However, in his quest for a nonproprietary data model for supporting an integrated process of Requirements and Configuration Management, the authors have found only one—which is, nonetheless, much more complex and comprehensive than the authors' desires and needs originally demanded: the GEIA-927 Common Systems Information Schema Concept (Colson 2005). The proposition of this work is, thus, to present a customized, open-source data model specific for supporting an integrated process of Requirements and Configuration Management according to the best practices of the literature and standards.

With a data model for integrated Requirements and Configuration Management it can be possible:

For Requirements Management:

- To specify all requirements and subrequirements of a project or program;
- To keep track of which requirements depend on which other requirements (interdependence tracking);
- To keep track of change history (who, when, why);
- To allow for change approval workflows;
- To allow for SQL queries and pivot-tables through Open Database Connectivity—or equivalent technologies—in Microsoft Excel or Google Sheets, to identify the impact of changes in requirements, track interdependencies, adherences, changes, and approvals;
- To allow for reuse of requirements between different projects;

For Configuration Management:

- To keep track of part-numbers, serial-numbers, suppliers, vendor warranties, drawings/diagrams in which the component is located, over its entire lifetime;
- To keep track of digital files: full-path, version, author, date, modifications;
- To keep track of what requirements—from those registered in the requirements management process—a component is addressed;
- To allow for queries via SQL and via pivot-tables in Microsoft Excel or Google Sheets;
- To document how a requirement is implemented in the final product.

RESULTS

In this section a Customized Data Model for Integrated Requirements and Configuration which can also be extended to Integrated Asset Management are defined and described.

The Proposed Model

The proposed model is based on the following two basic assumptions: 1) desires and needs from end-users or clients—which are called Stakeholders Requirements in the context of this work—, must be elicited into System Requirements; those requirements must be fulfilled by Configurations Items during the design phases; such items must finally be built into Assets during the assembly/integration/test/launch phase; 2) the main attributes of each one of the above entities must be kept and assessed in light of the original System Requirements and stakeholders desires and needs along all their life cycle. *Note*: The standard *IEC/IEEE 29148 – Systems and software engineering – Life cycle processes – Requirements engineering*, in its Section 6.2, "Stakeholder Requirement Definition Process," states that this process "identifies stakeholders, or stakeholder classes, involved with the system throughout its life cycle, and their needs, expectations, and desires" (ISO/IEC/IEEE 2011, p. 19). Also, Blanchard and Blyer (2016), in their book *System Engineering Management*, use extensively the term "desires" regarding the process of stakeholder's requirement discovery (Blanchard and Blyer 2016). That is the reason why, throughout this text, this author uses the expression "Desires and Needs – D/N" to denote those original aspirations.

In this section the main components of the model are described, and their main attributes and relationships explained. The model is composed of 30 entities—or classes. A picture of it may be downloaded in a zoomable format from: https://github.com/sergiorfoliveira/ircm/blob/master/IRCM%20UML%20Diagram.pdf.

In the following sections, a divide-and-conquer approach is implemented to discuss in detail the 6 main entities of the model. Its 24 other classes, considered auxiliary, are also briefly discussed. Anyhow, the complete model source-code is available for download at: https://github.com/sergiorfoliveira/ircm.

Interesting enough, the very model being proposed is, itself, subject to the assumptions listed in the previous paragraph. In fact, not only the model, but all other components that take part in this same undertaking.

Tables 1 and 2 list the main classes and auxiliary classes of the proposed model, respectively.

Table 1. The proposed model main classes.

Class	Short Description
Desires and Needs	A class for cataloging stakeholders' desires and needs
Requirements List	A class for cataloging requirements
Items	A class for cataloging CI and Assets
Examination Verification Validation Events	A class for cataloging assessment events
Part Numbers	A class for cataloging part-numbers
Asset Requirements List	A class for assigning requirements to CIs or Assets

Source: Elaborated by the authors.

Table 2. The proposed model auxiliary classes.

Class	Short Description
Asset Classes	A class for cataloging asset categories
Changes	A class for cataloging changes in model and items
Clauses List	A class for cataloging lists of S/R clauses
Configuration Baselines	A class for cataloging Configuration Baselines
Hazards	A class for cataloging hazards
Hazards List	A class for cataloging lists of hazards
Interfaces	A class for cataloging CI or Asset interfaces
Interface Types	A class for cataloging types of interfaces
Manufacturers	A class for cataloging Assets manufacturers'
Parent-Child 01	A class for "multiparent" D/Ns
Parent-Child 02	A class for "multiparent" requirements
Programs	A class for cataloging Programs (i.e., sets of Projects)
Projects	A class for cataloging Projects
Requirement Levels	A class for cataloging levels of requirements
Requirement Types	A class for cataloging types of requirements
Stages	A class for cataloging project stages
Stakeholders	A class for cataloging Stakeholders
Standards	A class for cataloging Standards/Regulations
Standard Clauses	A class for cataloging S/R clauses
Violations	A class for cataloging requirement violation reasons
VV Actions	A class for cataloging actions of assessment events
VV Methods	A class for cataloging methods of assessment events
VV Status	A class for cataloging Pass/Fail status
Yes No	A class for cataloging standard Yes/No assessment answers

Source: Elaborated by the authors.

The Desires and Needs Class

This is where all begins. This class is intended to keep all desires and needs from end-user clients or stakeholders. Its attributes are shown in Table 3. For many of these attributes the authors have been inspired by the American National Aeronautics and Space Administration's *Systems Engineering Handbook* (NASA 2007, p. 47-48); by the Chapter 5 from the book *Engineering Design: A Systematic Approach*, from Pahl *et al.* (2007, p. 145-158); by the work of Koelsch (2016, p. 265-271); and by their very own desires and needs.

Table 3. The Desires and Needs Class attributes.

Attribute	Description
GlobalID	This is a unique identifier for the desire/need
SetName	An optional attribute that allows grouping desires/needs into sets
Label	An optional label (e.g., 1.1, 1.2, 1.2.1, etc.)
ParentID	If the desire/need is derived from another, this attribute keeps its “parent id,” i.e., the desire/need from where it is derived
IsAlternativeForGlobalID	If the desire/need is a restatement of another, this attribute keeps the desire/need “id” from which it is a restatement
Type	Type of the desire or need (e.g., Safety, Security, etc.)
DesireOrNeedText	Statement of the desire or need
Rationale	Rationale of the desire/need
CreationTimestamp	The timestamp of the registration of the desire/need
CreatorStakeholder	The stakeholder who registered the desire or need
RequestorStakeholder	The stakeholder who manifested the desire or need
VerifierValidatorStakeholder	The stakeholder who will be in charge of validating the D/N
Frozen	Whether this desire/need is frozen or not
FreezerStakeholder	The stakeholder who has frozen the desire/need
FreezingTimestamp	The timestamp when the D/N is frozen

Source: Elaborated by the authors. Notes: If the D/N is a “child” D/N, i.e., it exists because a “parent” D/N has been broken into sub-D/Ns for clarity or convenience—then the attribute “ParentID” contains the “GlobalID” of the “parent” D/N. If the D/N has more than one “parent” —i.e., more than one originating D/N—, the model has an auxiliary class called Parent-Child 01, which relates the D/N with its other parents. More on this follows in the section where Parent-Child auxiliary classes are discussed. The “Type” attribute is useful for qualifying a D/N as being related to issues like “Cost,” “Quality,” “Safety,” “Security,” etc. This allows for keeping track of specific types of D/Ns. The “Frozen” attribute is intended to signal whether a D/N is still being revised or is already available for further elicitation into one or more System. The “IsAlternativeForGlobalD” attribute is intended to the tracking of changes of a D/N: Rather than deleting the originating D/N and replacing it for a new one, a restatement D/N keeps a pointer to the D/N from which it is derived, what allows for auditing and liability control. It is important to note the difference between an “alternative” D/N—which is basically a restatement from another one previously existent and intended to be replaced—, from a “child” D/N, which is a D/N that exists because a “parent” D/N had been broken into sub-D/Ns for clarity or convenience—this is the reason for the attribute “ParentID.” For both attributes—“Frozen” and “IsAlternativeForGlobalD”—the authors have been inspired by Chapter 30 of the book “System Engineering Analysis, Design, and Development” from Charles Wasson (2016, p. 652-653).

The Requirements List Class

Once a D/N has been frozen, it is ready to be further elicited into one or more System Requirements, and the class “Requirements List” is intended to keep all those elicited requirements. Its attributes are shown in Table 4.

Table 4. The Requirements List Class attributes.

Attribute	Description
ReqID	This is a unique identifier for the requirement
OriginatingDesireOrNeed	The GlobalID of the originating D/N
OriginatingRequirement	The ReqID of the requirement from which this requirement is derived, if any
ProgramRequirementOf	If it is a program requirement, then this attribute contains the Program Name, Otherwise, stays empty
ProjectRequirementOf	If it is a project requirement, then this attribute contains the Project Name, Otherwise, stays empty
CreationTimestamp	The timestamp of the registration of the requirement
SetName	An optional attribute for grouping requirements in sets
ReqLabel	An optional label (e.g., 1.1, 1.2, 1.2.1, etc.)
IsAlternativeOf	If the requirement is a restatement of another, this attribute keeps the requirement "id" from which it is a restatement
ReqType	Type of the requirement (e.g., Safety, Security, etc.)
ReqText	Statement of the requirement
Rationale	Rationale of the requirement
AssociationTimestamp	The date/hour when the requirement has been associated to the project or program
AssociatorStakeholder	The stakeholder who registered or associated the requirement to the project/program
VerifierValidatorStakeholder	The stakeholder who will be in charge of V&V activities for this requirement
ReqLevel	Stakeholder Requirement or System Requirement
Priority	The priority of that requirement for that project/program
Difficulty	The greater the number the greater the difficulty in implementing the requirement
StageOfAssociation	The stage when that requirement is associated to the project (e.g., "Concept and Technology Development," "System Assembly, Integration, Test and Launch" etc.)
StageToBeVV	Stage before or at which the requirement must be verified or validated
VVMethod	Verification/Validation method (e.g., "Demonstration," "Test," "Simulation", etc.)
Frozen	Whether this requirement is frozen or not
FreezerStakeholder	The stakeholder who has frozen the requirement
FreezingTimestamp	The timestamp when the requirement is frozen
HazardListName	The associated hazard list name
StdClausesListName	The associated standards clauses list name

Source: Elaborated by the authors. Notes: The "OriginatingDesireOrNeed" attribute points to the frozen D/N from which the requirement has been derived, if any. If the requirement is a "child" requirement—i.e., if it exists because a "parent" requirement has been broken into subrequirements for clarity or convenience—then the attribute "OriginatingRequirement" contains the "ReqID" of the "parent" requirement. If the requirement has more than one "parent"—i.e., more than one originating requirement—, the model has an auxiliary class called "Parent-Child 02," which relates the requirement with its other parents. More on this follows in the section where Parent-Child auxiliary classes are discussed. The attributes "ProgramRequirementOf" and "ProjectRequirementOf" indicate if the requirement refers to a program—i.e., a group of projects—or to a project that belongs to a program, and, if so, it contains the program name or the project name, as appropriate. The proposed model has two auxiliary classes—"Programs" and "Projects"—from which the names come from. The "SetName" attribute has two functions: to allow for grouping requirements in sets, and to allow for the assignment of many requirements—the entire set—to a Part Number or to a Configuration Item at once. More on that is discussed in the section about the Items class. The "ReqLevel" attribute keeps the level of the requirement, for instance, "Stakeholder Requirement" or "System Requirement." In the context of this work, a requirement which has not yet been reviewed considering the clause 5.2 from standard ISO/IEC/IEEE 29148:2011 – Requirements fundamentals, is still a Stakeholder Requirement, not yet a System Requirement. The proposed model has a class to keep track of these reviews. The "ReqType" attribute is useful for qualifying a Requirement as being related to issues like "Cost," "Quality," "Safety," "Security," etc. This allows for keeping track of specific types of requirements. The "Priority," "Difficulty" and "StageOfAssociation" attributes exist not only for convenience, but also for compliance with the clause 5.2 from standard ISO/IEC/IEEE 29148:2011 – Requirements fundamentals. The "StageOfAssociation" attribute allows values from an auxiliary class called Stages. The prototype application assumes usable values for this attribute inspired by the Chapter 6 of the document *SX000i – International Guide for the Use of the S-Series Integrated Logistics Support Specifications* (ASD/AIA 2016, p. 6.4), but the auxiliary class Stages allows for other stage names. The "VVMethod" attribute exists mostly for compliance with clause 6.2 from standard ISO/IEC/IEEE 29148:2011—The Stakeholder requirements definition process—, although it allows for the inclusion of other methods. The prototype application includes a method called "Examination" not for verification/validation actions, but for requirement reviews (i.e.: the assessment of the quality of the requirement itself). The "HazardListName" attribute allows for the assignment of a list of previously identified hazards which the requirement aims to mitigate. The proposed model has two auxiliary classes called Hazards and Hazards List which allow for keeping identified hazards and groups of hazards, respectively. This attribute has been conceived to work in conjunction with the attribute "ReqType" for requirements of type "Safety." The "StdClausesListName" attribute allows for the assignment of a list of previously identified standard/regulation clauses on which the requirement has been based, if any. The proposed model has three auxiliary classes called Standards, Standards Clauses and Clauses List, which allow for keeping standards/regulations, the clauses belonging to those standards/regulations, and groups of clauses, respectively. This is useful for further documenting the reason of existence of a requirement, and to map standard/regulation clauses into requirements or vice versa.

The Items Class

Along with the Desires and Needs class and the Requirements List class, the Items class is among the top three main classes of the proposed model since it allows for keeping track of Configuration Items and Assets. Its attributes are shown in Table 5.

Table 5. The Items Class attributes.

Attribute	Description
ID	This is a unique identifier for the Asset or the Configuration Item
Name	This is the name of the item
ConfigurationBaseline	This is the name of the configuration baseline to which the item belongs, if any
SerialNumber	If the item is a real asset—e.g., a built/purchased item—, this attribute may keep the serial number of the asset—or version number, if the asset is a document. Otherwise, it stays empty
Tag	This is an optional attribute for keeping the tag of a built item—e.g., the registration of an aircraft
PartNumber	The part-number of the asset. It must contain a valid part-number previously cataloged in the Part Numbers class
VendorNickname	The nickname of the vendor of the asset, if it is a purchased item. Otherwise, it stays empty
OwnerNickname	The nickname of the owner of the asset, if it is a purchased item. Otherwise, it stays empty
Count	The count—i.e., the number—of this item in its parent item or in the current configuration baseline
ParentID	The "parent" item, if the asset is component of a built/purchased item already cataloged. Otherwise, it stays empty
CreatorStakeholder	The stakeholder who cataloged the item
CreationTimestamp	The timestamp when the item has been cataloged
FQFN	It may contain the fully qualified filename, if the item is an electronic document, or the specification document/drawing, if the item is not an electronic document.
Note	Some description of the item
RequirementsSetName	It may point to the attribute "SetName" of the Requirements List class, in case this item must comply with a given set of already-cataloged requirements

Source: Elaborated by the authors. Notes: If the proposed model is used in a design organization, the attribute "ConfigurationBaseline" may be used to keep track of the baseline name or version name of the configuration being designed, according to standard IEEE 828-2012. If the proposed model is being used in a non-design organization—for instance, in an end-user or client organization—the attribute "ConfigurationBaseline" may be left unused. Although an item may not have a serial number—for instance, items not yet built do not have serial numbers—, the proposed model assumes that every item must have a "PartNumber"—be it an internally-assigned part-number, if the model is being used in a design organization—, or a manufacturer-assigned part-number, if the model is being used in a non-design organization, for instance, in an end-user or client organization. The proposed model has a class called Part Numbers which keeps track of usable part-numbers. The proposed model is agnostic in relation to serial number and part-number formats. For items purchased from other organizations, the attribute "VendorNickname" may contain the name of the vendor. The proposed model does not have a class for vendors since this information can be obtained from other sources or models. If the item does not belong to the organization where the proposed model is used, for instance, if the organization only maintains the requirements of the item, then the attribute "OwnerNickname" may contain the organization name of the item owner. The attribute "ParentID" allows for keeping track of items that are subitems of another already cataloged item. The attribute "RequirementsSetName" may point to the attribute "SetName" of the Requirements List class, in case the item must comply with a group of requirements. The proposed model supports another way of assigning individual requirements to individual items.

The Examination/Verification/Validation Events Class

Along with the Asset Requirements List class, this is one more class that establishes relations between requirements and items. The rationale is: if there is a requirement, then there must be at least 3 events related to that requirement, each event having one action, such as "Examination," "Verification" or "Validation", in the following order:

- An assessment or review of the quality of the written requirement, considering the clause 5.2 from standard ISO/IEC/IEEE 29148:2011 – Requirements fundamentals. In this proposed model, this kind of action is called "Examination," but an organization adopting this model may call that by any other name;
- A "Verification" event, in which the designed or built system/subsystem/element is verified against the requirement to answer the question: was the item designed/built correctly? There are many possible ways for performing verification assessments—for instance "Analysis," "Demonstration," "Inspection," "Test" and "Simulation" (Buede 2009, p. 64). A prototype application has already these methods cataloged and allows for the addition of others and the modification/deletion of existing ones.

- A “Validation” event, in which the designed or built system/sub-system/element is validated by the requestor stakeholder to answer the question: Was the correct item designed/built? *Note*: This is a final assessment, at the real operational environment, without the use of computational simulation.

Naturally, during the design/build phases of an Item there may exist many iterations of these 3 types of events, until a requirement is considered to “pass” its “Examination,” or until a tuple Item-Requirement is considered to “pass” its “Verification” and “Validation” events. And after that—and all over the entire life cycle of an item—, the Item may be subject to many “Verification” and “Validation” events, depending on modifications it suffers. This class allows for recording all those events.

The attributes of this class are shown in Table 6.

Table 6. The Examination/Verification/Validation Events Class attributes.

Attribute	Description
ID	This is a unique identifier for the event
ReqID	This attribute points to the requirement unique identifier which has been the object of this event
AssetID	If the action of event is “Verification” or “Validation,” then this attribute points to the item unique identifier which is the object of the verification or validation event. Otherwise, if the event action is “Examination,” this attribute stays empty because what has been assessed is the quality of the requirement itself, not the compliance of an item to a requirement
EventAction	The action that has been performed in the event. For instance: “Examination,” “Verification,” “Validation”
EventTimestamp	The timestamp when the event took place
EventMethod	The method used to perform the event action. For instance: “Analysis,” “Demonstration,” “Inspection” etc
EventStatus	The resulting status of the action. For instance: “Pass,” “Fail”
MainViolation	If the requirement has failed the event, this attribute keeps the primary reason of violation. For instance, “Non-compliant,” “Ambiguous,” “Non-complete,” “Non-consistent,” etc.
ExecutorStakeholder	The stakeholder who has performed the event action
EventDescr	A text attribute for describing event details, if desired

Source: Elaborated by the authors. Notes: The proposed model assumes that never an event is deleted from the events repository. In case of a “Fail” event, a future “Pass” event will have to exist regarding the same item and/or requirement. The prototype application assumes as possible values for the “MainViolation” attribute those cited in the clause 5.2 from standard ISO/IEC/IEEE 29148:2011 – Requirements fundamentals (ISO/IEC/IEEE 2011, p. 6-18). The model has an auxiliary class called Violations, which may contain other reasons beyond those. A prototype application created by the authors assumes as possible values for the “EventMethod” attribute those listed in the book “The Engineering Design of Systems: Models and Methods,” from Dennis Buede (2009, p. 64), except for the “Examination” method, for which the authors have been inspired by the definition 4.1.23 from ISO/IEC/IEEE 29148:2011 (p. 6): “confirmation by examination that requirements (individually and as a set) are well formed.” The model has an auxiliary class called Verification and Validation Methods, which may contain other methods beyond those.

The Part-Numbers Class

As already stated, the proposed model assumes that an Item may not have a serial number, but every single item must have a “PartNumber”—be it an internally-assigned part-number, if the model is being used in a design organization—or a third-party, manufacturer-assigned part-number, if the model is being used in a non-design organization, for instance, in an end-user or client organization. The attributes of this class are shown in Table 7.

Table 7. The Part-Numbers Class attributes.

Attribute	Description
OurPartNumber	This is a unique identifier of the part-number
OurDescription	This attribute may contain a description of the part-number
ManufacturerNickname	This attribute may contain the nickname of the manufacturer of the item, if it is a third-party item. Otherwise, it stays empty
MafacturerPartNumber	This is a unique identifier of the part-number, in case it is a third-party item. Otherwise, it stays empty
ManufacturerDescription	This attribute may contain a description of the manufacturer, in case it is a third-party item. Otherwise, it stays empty
Class	This attribute contains the class to which the part-number belongs. For instance: “1510 – Aircraft – fixed wing,” “1520 – Aircraft – rotary wing,” “1540 – Gliders,” “1550 – Drones,” etc. The informed class must be previously cataloged by the Class auxiliary class
IsAlternativeFor	If this part-number is an alternative for another part-number already cataloged, this attribute points to its alternative unique identifier, i.e., its “OurPartNumber” attribute
RequirementsSetName	It may point to the attribute “SetName” of the Requirements List class, in case this part-number must comply with a given set of already-cataloged requirements

Source: Elaborated by the authors. Notes Regarding the attribute “Class,” the proposed model has an auxiliary class for containing usable categories of part-numbers. In the prototype application are proposed some part-number categories inspired in the U.S. Federal Supply Classes (U.S. Department of Defense 2018), but the model has an auxiliary class called Asset Classes which may contain other categories beyond those. The attribute “RequirementsSetName” may point to the attribute “SetName” of the Requirements List class, in case the part-number must comply with a group of requirements. It is assumed that all items having that part-number will have to comply with those requirements. But the proposed model supports another way of assigning individual requirements to part-numbers. The proposed model is agnostic in relation to part-number formats.

The Asset Requirements List Class

As already stated, the proposed model provides two ways of assigning requirements to items or to part-numbers: a) through the “SetName” attribute of the Requirements List class—and its counterpart “RequirementsSetName” attribute of the Items and Part Numbers classes; b) through the Asset Requirements List class, which allows for assigning individual requirements either to individual items or to individual part-numbers, using the Asset Requirements List class. The attributes of this class are shown in Table 8.

Table 8. The Asset Requirements List Class attributes.

Attribute	Description
ID	This is a unique identifier of the assignment
ListName	This is the name of the list of assignments
AssetID	If the requirement is being assigned to an item, this attribute contains the unique identifier of the item, previously cataloged through the Items class. Otherwise, it stays empty
PartNumberID	If the requirement is being assigned to a part-number, this attribute contains the unique identifier of the part-number, previously cataloged through the Part Numbers class. Otherwise, it stays empty
ReqID	This is the unique identifier of the requirement being assigned to an item or to a part-number

Source: Elaborated by the authors. Notes: It is assumed that if a requirement set is assigned to a part-number—through its “RequirementsSetName” attribute—, all items that share that part-number will have to comply with that group of requirements, even though the tuple item-requirement is not related by the Asset Requirements List class. This allows for a powerful way of assigning multiple requirements to multiple items without the need of individually assigning each requirement to each item. It is assumed that for each assignment done through the Asset Requirements List class, either a value to the attribute “AssetID” or to the attribute “PartNumberID” will be informed, not to both.

Auxiliary Classes

The proposed model has 30 main classes. The other classes—some of them already cited in previous Sections—, are briefly listed below. The auxiliary classes are:

- Asset Classes: Allows for keeping asset categories—e.g., “1510 – Aircraft – fixed wing,” “1520 – Aircraft – rotary wing,” “1540 – Gliders,” “1550 – Drones,” etc. The prototype application proposes some usable categories based on U.S. Federal Supply Classes (U.S. Department of Defense 2018), but the Asset Classes may deal with other categories beyond those.
- Changes: Allows for keeping changes made in the model itself and in the values kept by its main classes. This is a true auxiliary class—or meta-class—since nothing else in the model depends on this class.
- Clauses List: Allows for keeping groups of clauses taken from regulations or standards. These groups—or lists—of clauses may then be assigned to requirements. This class is directly related to the Standards and Standard Clauses classes.
- Configuration Baselines: Allows for keeping attributes of configuration baselines, such as whether the baseline is frozen or not, and if so, who performed the freezing of the configuration, and when.
- Hazards: Allows for keeping hazards, e.g., “There may be child requirements that are not compliant.”
    - Hazards List: Allows for keeping groups of hazards kept by the Hazards class. These groups—or lists—of hazard may then be assigned to requirements.
    - Interfaces: Allows for keeping how many items having a specific part-number may be contained in—or connected to—other items having other specific part-numbers through specific types of interfaces. This is useful for applications that want to verify if a configuration being designed is valid.
    - Interface Types: Allows for keeping usable interface types, e.g., “ARINC-629,” “ARINC-664-P7-AFDX,” etc. This class is directly related to the Interfaces class.
    - Manufacturers: Allows for keeping attributes of usable manufacturers, such as the nickname and the “id” of the manufacturer in the organization ERP system. By design, it has been chosen to keep this class as minimalistic as possible in the proposed model, since such attributes are typically maintained by other models of an organization.
    - Parent-Child 01: Allows for keeping D/Ns that have more than one “parent” D/N.
    - Parent-Child 02: Allows for keeping requirements that have more than one “parent” requirement. *Note*: Even though the model allows for requirements with more than one “parent,” this does not mean this functionality must be used by an adopting Organization. This is a discretionary functionality. For example, a professional application based on this model may be configured to enforce the limit of one “parent”, at most, for any requirement. However, in some circumstances it may be desirable, or even mandatory, to have a requirement with more than one “parent,” depending on the Organization policy and/or the system being designed. Another reason for including that functionality is that at least one author—e.g., Robert Halligan (2018)—who was a reviewer of EIA 632 and EIA 731 standards and a content contributor to EIA 632 standard—, have manifested a favorable opinion about the possibility of having multiple “parent” requirements to a single “child” requirement.
    - Programs: Allows for keeping attributes of existing programs. *Note*: In this context, a program is a group of projects. By design, it has been chosen to keep this class as minimalistic as possible in the proposed model, since such attributes are typically maintained by other models of an organization. This class is directly related to the Projects class.
    - Projects: Allows for keeping attributes of existing projects. By design, it has been chosen to keep this class as minimalistic as possible in the proposed model, since such attributes are typically maintained by other models of an organization.
    - Requirement Levels: Allows for keeping the level of a requirement, for instance, “Stakeholder Requirement” or “System Requirement.” In the context of this work, a requirement which has not yet been reviewed considering clause 5.2 from standard ISO/IEC/IEEE 29148:2011 – Requirements fundamentals – is still a Stakeholder Requirement, not yet a System Requirement. Anyhow, this class supports any values one may find appropriate to use.
    - Requirement Types: Allows for keeping usable types of requirements, such as “Cost,” “Quality,” “Safety,” “Security,” etc. The prototype application proposes some usable values for this attribute based on the following sources: the book

Engineering Design - A Systematic Approach from Pahl *et al.* (2007, p. 567) and the clause 5.2.8.2 from the standard ISO/IEC/IEEE 29148:2011 (p. 13).

- Stages: Allows for keeping usable stages—or phases—for program and project life cycles, such as "Pre-Phase A: Concept Studies," "Phase A: Concept and Technology Development," "Phase B: Preliminary Design and Technology Study," etc. The prototype application proposes some usable values for this attribute based on the document *SX000i - International Guide for the Use of the S-Series Integrated Logistics Support Specifications* (ASD/AIA 2016, p. 6.4).
- Stakeholders: Allows for keeping stakeholders' attributes. By design, it has been chosen to keep this class as minimalistic as possible in the proposed model, since such attributes are typically maintained by other models of an organization.
- Standards: Allows for keeping standards/regulations attributes, such as title, issuer and year.
- Standard Clauses: Allows for keeping relevant clauses from standards/regulations. This class is directly related to the Standards and Clauses List classes.
- Violations: Allows for keeping reasons of violation of requirements, such as "Ambiguous," "Non-complete," "Non-consistent," etc. The prototype application proposes some usable values for this attribute based on the clause 5.2 from standard ISO/IEC/IEEE 29148:2011 (p. 8-14).
- VV Actions: Allows for keeping actions related to verification/validation events, such as "Verification," "Validation," "Examination," etc. The prototype application proposes some usable values for this attribute based on definitions 4.1.22, 4.1.23, 4.1.32, 4.1.33 from standard ISO/IEC/IEEE 29148:2011 (p. 6-7).
- VV Methods: Allows for keeping methods related to requirement review/verification/validation events, such as "Analysis," "Demonstration," "Inspection," "Test," etc. The prototype application proposes some usable values for this attribute based on the following source: the book "The Engineering Design of Systems: Models and Methods," from Dennis Buede (2009, p. 64), except for the "Examination" method, which is based in the definition 4.1.23 from ISO/IEC/IEEE 29148:2011 (p. 6).
- VV Status: Allows for keeping the result of a requirement review/verification/validation event, such as "Pass" or "Fail."
- Yes No: Allows for keeping standard responses to questions like "Have the requirement passed the assessment?" *Note*: the "Yes No" class is useful for customizations and for localization, for example, to adapt some instantiation of model to languages different from the English. This is suitable in Organizations where the words "Yes" and "No" must be substituted for their respective counterparts in their native languages. The same may be said about other classes—for instance, the "VV Status" class, which may contain words like "Pass" or "Fail" etc.

DISCUSSION

Benefits of the Proposed Model

The proposed model might lead to the following benefits:

- It allows to build a software application for integrated Requirements and Configuration Management, since the support for necessary referential integrities between configurations, requirements, assessment events, part-numbers, stakeholders and manufacturers is built into the model.
- More than that, it enables to extend such a software application to integrate Asset Management into Requirements and Configuration Management, since the model already includes the support for referential integrities between serial numbers, part-numbers, configurations, vendors and manufacturers, allowing the application to keep track of requirement compliance over the entire life cycle of an asset.
- It enables a software application to implement the management of requirements, configurations and assets in an integrated manner as preconized by Systems Engineering disciplines.
- The changes in a requirement itself may be tracked from the original D/N over the entire life cycle of the requirement, since the model allows for requirement restatement without the exclusion of old versions.

- Assessment events—e.g., examination, verification and validation events—, in which items are verified/validated in light of requirements, and requirement statements are examined in light of standards—, may be easily tracked over the entire life cycle of the requirements and over the entire life cycle of the item, since the model allows for newer events to be included without the exclusion of older events. Among other things, this enables to track who are the authors that write the best and the worse requirement statements, as a support information for training planning.
- Standards and regulations can be integrated into requirements by a software application since the necessary attributes and support for referential integrities are already built into the model. Also—and for the same reason—, hazards can be easily integrated into requirements.
- Certain requirements, like the ones regarding to safety, can be classified and tracked separately from other types, as well as the related hazards which are relevant to those requirements, since the model already has attributes for keeping the type of the requirement and the hazards.
- Since the model is all-relational, non-hierarchical, and non-object-oriented, it can be implemented in a conventional Relational Database Management System.
- It allows for tracking in which stage of the project a requirement has been created or restated, and who did that;
- It can be used as a teaching aid tool in classes related to Requirements and Configuration Management.
- It allows for keeping statistics or key performance indicators on the quality of the requirement writing process, for liability control, for training plans, and to assess the impact of requirements volatility on the Systems Engineering discipline.
- It can be used as a support tool in the QFD methodology, enabling the incorporation of "the voice of the regulator" to the QFD Matrix I.

All the basic attributes necessary for implementing an application with approval workflow functionality are already included in the proposed model. Thus, a new requirement or configuration—or a change in an existing requirement or an existing configuration—, can be easily checked against requirements, standards and decision-making policies for compliance verification and approval.

The verification of the above benefits—except for the last four, which are just inferred—was carried out by a prototype web-based application which is available in the GitHub repository. The validation of all the above benefits and the verification of last four ones are left for future works.

The authors understand that an open data model such as the one proposed, compliant with the System Engineering taxonomy, allows for both—the analysis of safety issues in the design phase, and the deployment of processes for minimizing the chance of neglecting safety-relevant aspects during the development of critical systems; not to mention other aspects such as security, quality, etc.

This would be particularly useful for small and medium Organizations and for the academic community—that may not be willing to acquire a sophisticated, proprietary market solution, configure it, and train their staff to effectively use it to implement basic Systems Engineering practices—, and to those who would be interested in extending or adapting the model for their specific needs.

CONCLUSION

From the ground up, the process of designing the data model has been done in light of the fact that a proposed model originated from this work would have to fulfill, itself, a set of requirements and would have to result in configuration items and assets which should be, themselves, managed, in an integrated way, by a prototype web application built to use the very proposed model, instantiated in the form of a relational database in an open-source RDBMS. If the originating requirements were robust enough, detailed enough and open enough, the proposed model could be verified and validated by third parties in light of its own requirements and/or third-party requirements and in light of the best practices of Systems Engineering, Requirements Management and Configuration Management disciplines.

The endeavor has been successful, resulting in a customized data model capable of providing the necessary support for an integrated process of Requirements and Configuration Management, and in a database and a prototype web application capable of verifying the model.

The result of this work contributes to the diffusion and incorporation of Systems Engineering best practices specially in academy and in small and medium-sized companies since they do not have the time, accumulated experience, and money available to implement from scratch SE processes.

AUTHORS' CONTRIBUTIONS

Conceptualization: Oliveira SRF; **Methodology:** Rocha GC; **Software:** Oliveira SRF; **Validation:** Rocha GC; **Formal analysis:** Oliveira SRF; **Investigation:** Oliveira SRF; **Resources:** Oliveira SRF; **Data Curation:** Oliveira SRF; **Writing - Original Draft:** Oliveira SRF; **Writing - Review & Editing:** Oliveira SRF; **Visualization:** Oliveira SRF; **Supervision:** Rocha GC; **Project administration:** de Andrade D.

DATA AVAILABILITY STATEMENT

The data are available in: https://github.com/sergiorfoliveira/ircm

FUNDING

Not applicable.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[ASD/AIA] AeroSpace and Defence Industries Association of Europe, Aerospace Industries Association (2016) SX000i - International Guide for the Use of the S-Series Integrated Logistics Support (ILS) Specifications, Brussels: ASD/AIA. [accessed Dec 9 2017]. http://www.sx000i.org/docs/SX000i_Issue_1.1.pdf

Blanchard BS, Blyer JE (2016) System engineering management. 5th ed. Hoboken: Wiley.

Bone M, Cloutier R (2010) The current state of model based systems engineering: Results from the OMG™ SysML Request for Information 2009. Paper presented 8th Conference on Systems Engineering Research. CSER 2010; Hoboken, NJ, USA. [accessed May 31 2022]. https://www.omgsysml.org/SysML_2009_RFI_Response_Summary-bone-cloutier.pdf

Bonnet S, Voirin J-L, Normand V, Exertier D (2015) Implementing the MBSE cultural change: Organization, coaching and lessons learned. INCOSE International Symposium 25(1):508-523. https://doi.org/10.1002/j.2334-5837.2015.00078.x

Buede DM (2009) The engineering design of systems: Models and methods. 2nd edition. Hoboken: Wiley.

Colson J (2005) GEIA-927 Common data schema for complex systems. Huntsville: U.S. Army, Logistics Support Activity.

Department of Defense (US) (2018) Defense Logistics Agency. ASSIST Quick Search Help. [access Jan 29 2018]. http://quicksearch.dla.mil/qsHelp.aspx

Halligan R (2018) Multiple parents to a single child requirement. Project Performance International. [accessed Feb 19 2018]. https://www.ppi-int.com/resources/systems-engineering-faq/multiple-parents-single-child-requirement/

[IEEE] Institute of Electrical and Electronics Engineers (2012) IEEE 828:2012: IEEE Standard for Configuration Management in Systems and Software Engineering. New York: IEEE.

[INCOSE] International Council on Systems Engineering (2017) History of systems engineering. [Accessed Dec 07 2017]. https://www.incose.org/AboutSE/history-of-systems-engineering

[ISO/IEC/IEEE] International Organization for Standardization / International Electrotechnical Commission / Institute of Electrical and Electronics Engineers (2011) ISO/IEC/IEEE 29148 – Systems and software engineering – Life cycle processes – Requirements engineering. Geneva: ISO/IEC/IEEE. [access Dec 9, 2017]. https://www.iso.org/standard/72089.html

Jänsch J, Birkhofer H (2006) The development of the Guideline VDI 2221 – The change of direction. Paper presented DS 36: Proceedings DESIGN 2006, The 9th International Design Conference, Dubrovnik. [access Dec 9, 2017]. https://www.designsociety.org/publication/18983/THE+DEVELOPMENT+OF+THE+GUIDELINE+VDI+2221+-+THE+CHANGE+OF+DIRECTION

Koelsch G (2016) Requirements writing system engineering. New York: Apress.

[NASA] National Aeronautics and Space Administration (US) (2007) Systems Engineering Handbook. Washington (DC): NASA.

[OMG] Object Management Group (2017) About the OMG System Modeling Language Specification Version 1.5. [accessed Dec 09 2017]. http://www.omg.org/spec/SysML/1.5/

Pahl G, Beitz W, Feldhusen J, Grote H (2007) Engineering Design – A systematic approach. 3 ed. London: Springer.

Ramos AL, Ferreira JV, Barceló J (2012) Model-based systems engineering: An Emerging Approach for Modern Systems. IEEE Trans Syst Man Cybern Syst 42(1):101-111. https://doi.org/10.1109/TSMCC.2011.2106495

Schlager KJ (1956) Systems engineering-key to modern development. IRE Trans Eng Manage EM-3(3):64-66. https://doi.org/10.1109/IRET-EM.1956.5007383

Software Engineering Research Group (2012) Survey on requirements engineering (RE) tools. Grupo de Ingeniería del Software. [accessed Dec 23 2017]. http://www.um.es/giisw/EN/re-tools-survey

Thales (2018) How Capella Help You? [accessed May 31 2018] http://polarsys.org/capella/index.html

[VDI] Verein Deutscher Ingenieure (2017) VDI-Standard: VDI 2221 Systematic approach to the development and design of technical systems and products. [accessed Dec 08 2017]. http://www.vdi.eu/guidelines/vdi_2221-methodik_zum_entwickeln_und_konstruieren_technischer_systeme_und_produkte/

Wasson C (2016) System engineering analysis, design, and development: Concepts, principles, and practices. Hoboken: Wiley.