A Semantic Driven Approach for Consistency Verification Between Requirements and FMEA

Gabriella Gigante, Francesco Gargiulo, Massimo Ficco and Domenico Pascarella

Abstract Consistency within the system life cycle is difficult to guarantee, due to the cross of different skills and requirements, often expressed by means of different languages. In particular, in safety-critical systems consistency between software requirements and safety analysis requires checks to guarantee that safety engineer needs are feasible and implemented by the system. Failure Mode and Effects Analysis (FMEA) is a systematic technique to analyze the failure modes of components, evaluating their impact and their mitigation actions, which are procedures to be implemented by operators or by the system itself (usually by the software). Although the actual efforts to centralize system information in a structured way, safety analysis is not tied in a structured manner to other systems, in particular to software. This paper proposes an automatic approach to check consistency between FMEA and software requirements with a bit effort of formalization. The approach models FMEA and software requirements with Resource Description Framework (RDF) triplets and checks their consistency on the basis of consistency rules.

Keywords Requirements engineering · Requirements verification · Consistency · RDF · Semantic distances · Ontologies

G. Gigante (✉) · F. Gargiulo · D. Pascarella
CIRA (Italian Aerospace Research Centre), Capua, Italy
e-mail: g.gigante@cira.it

F. Gargiulo
e-mail: f.gargiulo@cira.it

D. Pascarella
e-mail: d.pascarella@cira.it

M. Ficco
Department of Industrial and Information Engineering,
Second University of Naples, Aversa, Italy
e-mail: massimo.ficco@unina2.it

P. Novais et al. (eds.), *Intelligent Distributed Computing IX*,
Studies in Computational Intelligence 616,
DOI 10.1007/978-3-319-25017-5_38

1 Introduction

Consistency and completeness represent some of the main quality factors to design a "dependable system": inconsistent work-products imply system failures and unexpected behavior. In evolving software systems, consistency seems hard to maintain and to verify [2]. In safety-critical systems, consistency is hard to guarantee at process level due to the different skills to be provided.

Model-based development allows to define the system at different levels of abstraction, improving the consistency, the correctness and the completeness of its functional aspects. On the other hand, non-functional aspects such reliability and safety are more difficult to integrate. Safety standards require analysis during the system development [9–11], which provides a further view of the system by means of different formalisms. These dimensions of heterogeneity can easy lead to inconsistencies. In this context, providing evidence of consistency among the different work-products is challenging [13].

System safety analysis often are not linked to system software, which is usually in charge of recovery actions. These are identified in Failure Mode and Effects Analysis (FMEA) and define further software safety functions. Thus, software requirements model has to be consistent and complete against the FMEA model.

The idea discussed in this paper investigates a simple approach to verify inconsistencies between requirements and FMEA, both expressed in natural language. It proposes to model both requirements and FMEA by means of RDF (Resource Description Framework) triplets and to check consistency by means of empiric rules on similarity measures between inconsistent triplets [14].

2 Background

2.1 Consistency Concept

Engineering communities have built a conceptual framework for inconsistencies in system models, which proposes inconsistency management as a process. In literature, two general frameworks have been proposed: one by Finkelstein and one by Nuseibeh [5]. The activities of the frameworks are: detection of overlaps; detection, diagnosis, handling and tracking of inconsistencies; specification and application of a management policy for inconsistencies [6, 14]. Such activities have been studied at different levels. At process level, consistency is checked to hold between different models: it is horizontal if the models to check have the same abstraction level, otherwise it is vertical. At single level, consistency is checked within an artifact and is referred as internal or evolution consistency [4].

Any idea of consistency starts from the intuitive concept of contradiction: it denotes any situation in which a relationship $\mathfrak{R}$, that should hold between models or elements of a model, is found not to hold. Reference [3] defines inconsistency as

"*any situation in which two descriptions do not obey some relationship that should hold between them*". Although such definition gives rise to a large debate, it remains a reference starting point. In this way, the problem of detection can be completely addressed by a robust set of consistency rules and by a good algorithm, that shall browse the model and check the rules violation. Consistency rules can be simple when they refer to notations, development and local contingencies, wherein contradiction is a direct refutation of previously stated presented information. Moreover, information within the software models can be refuted in an indirect manner. A given set of facts could establish a potential situation that would contradict other facts within the models. Therefore, establishing consistency within a software model and between different system models is also a semantic task.

2.2 FMEA

FMEA is one of the first systematic techniques for dependability analysis, but it is still very popular throughout the safety-critical domains. It is a single point of failure analysis for safety-critical systems [12]. It examines the consequences of potential failures on the functionality of a system. Typically, FMEA is practiced on physical systems and the considered failure modes are the failures of physical components. More recently, different kinds of FMEAs are used in many domains to analyze general safety-critical systems (also software or processes) and can either be qualitative or quantitative [13]. FMEA goes through the following steps: identification of the system boundaries; identification of each function and associated component; identification of the potential failures mode for each function/component; identification of the impact of each failure at component level and at system level; allocation of the probability of each failure (only for quantitative evaluations). Finally, in the case of quantitative evaluations an additional step consists in allocating the probability of a failure for the associated component.

2.3 Internal and External Inconsistencies of FMEA

FMEA uses natural language to describe failure modes, failure effects and mitigation actions. It usually contains hundreds of lines, filled by a team of engineers that can be different from the development team, especially from the software team. In this sense, the FMEA internal and external consistency cannot be easily guaranteed. Failure modes should be the same type for each function and for each class of components. They should be described by means of the same terms, otherwise the evaluation of occurrence probabilities is difficult. Furthermore, if a new failure mode is discovered during the design, it should be added, resulting in a time consuming and error prone operation. Failure effects should be consistent with the corresponding failure modes. But if they are described in different ways, such check is difficult.

The choice of a seamless modeling framework could be the proper solution to such problems. Recent researches highlight difficulties to conceal system functional and not functional properties. On the one hand, research effort focuses on proposing unified approaches to system modeling by defining a comprehensive formalism. The drawback of this approach is the difficulty to put into practices, as the development team has to learn a new language [15]. On the other hand, research proposes a multi-formalism approach, which models each aspect of the system using the proper formalism and combines the different models for the overall system model by means of powerful operators. The main problems are the operators and the consistence between different formalisms.

Proposals of knowledge-based approaches, such as taxonomy or ontologies to automatically define FMEA, are the scope of recent studies. Such techniques could enforce the internal consistence, but the external consistence is still not covered. In this paper, the external consistency between FMEA actions demanded to software and software requirements is checked, being both specified in natural language.

3 Consistency Verification Framework

3.1 The Approach

The proposed approach adopts RDF to model each system artifact written in natural language. It evaluates semantic distances between RDF triplets by using similarity measures on the basis of referring ontologies. The semantic distances are checked according to thresholds inferred from a previous study, proposing a high level classification of inconsistency in order to derive general consistency rules with a confident level of possible inconsistencies coverage [14]. Finally, it experiments the defined rules to check inconsistency between a part of system FMEA and a set of software requirements with known inconsistencies.

3.1.1 The RDF Triplet Extraction

The use of RDF triplets allows to move to a more structured representation of the artifacts. This representation is not semantically equivalent in strict sense, but the key concepts are captured. Before the extraction, some well-known NLP (Natural Language Process) tasks would be executed, such as: lemmatization, NER (Named Entity Recognition) both for common entities (i.e., people, places and organization) and domain specific entities; compound words, abbreviations or acronyms detection; the word sense disambiguation, etc. These tasks often rely on one or more general purpose or domain specific ontologies. The *subj*, *pred* and *obj* are not plain words but they represent concepts of an ontology. In our approach triplets are identified according to the following rules:

- the passive form is translated into the active form;
- the complement of specification related to the subject is treated as a single entity, with the subject that must be present in the domain ontology;
- the complement of specification related to the object is treated as a single entity, with the object that must be present in the domain ontology;
- the complements of term, of agent, of half, of time, of motion are translated defining a triplet for each of them and associating to each verb the preposition they imply (for example, the predicate and the term complement are translated in *predicate_TO*, *term complement*);
- if the requirement expresses a conditioned action, which is an action that should be performed only if a condition is true, the triplets are at least 3: the first asserting the action with the object (main) A, the second expressing the precondition B, the third <B, precondition, A> expressing that A is executed only if B is true;
- if the requirement expresses a constrained action, that is an action that should be performed only if a constraint is true, the triplets are at least 3: the first asserting the action with the object (main) A, the second expressing the constraint B, the third < *B*, constraint, A> expressing that A is executed only if B is true;
- FMEA mitigation actions are translated as conditioned actions, where precondition triplets are built up with the failing component as subject, the verb "*fails*" as predicate, the conjunction of failure mode and the phase to which FMEA refers as object.

In the above definitions, we use the terms precondition and constraint pointing out two different concepts. Precondition is a requirement that depends on the specific system, mission or operation, and it is extracted from the artifacts. Constraint expresses a necessary condition due to physical issues (not depending on the application) and it is provided by domain knowledge.

The extraction of precondition triplets has been implemented manually.

3.1.2 The Adopted Similarity Measure

First of all, it is necessary to identify a referring ontology to calculate similarity measure. We adopted Wordnet 2.1 [7, 8] and some libraries of Wordnet 3. We model some general main concepts of the domain under study by RDF triplets in a text file. We choose the edge-based Wu and Palmer [1] similarity measure:

$$WP(x, y) = \frac{2p(lcs)}{p(x) + p(y) + 2p(lcs)}, \tag{1}$$

where *lcs* is the last common subsumer of x and y and $p(x)$ and $p(y)$ are the levels of x and y in the wordnet tree. To overcome the problem of similarity measure calculated on non-homogenous ontologies, we adopt the same technique proposed in [14], extending the upper ontology by simply "attaching" the domain specific concepts to

the common root concept. For example, to calculate the distance between ground software (concept of the domain ontology) and system, Eq. (1) becomes:

$$WP(\text{“}ground\ software\text{”}, \text{“}system\text{”}) = \frac{2p(lcs)}{p(\text{“}ground\ software\text{”}) + 1 + p(\text{“}system\text{”}) + 2p(lcs)} \quad (2)$$

3.1.3 The Inconsistencies Rules

FMEA and software requirements are consistent if the following conditions hold: every FMEA action assigned to software is a software function implemented under a precise precondition, exactly the same of FMEA triplet precondition; no software function contradicts or obstacles any FMEA action, that is, software requirements are internally consistent. Such considerations led to the adoption of a subset of the inconsistency rules proposed in previous paper [14], mainly related to conflicting actions or conflicting preconditions.

Let us assume the RDF triplet $t_i = <s_i, p_i, o_i>$ and $T_i = <t_i>$ the set of triplets under study. To detect inconsistent triplets in the identified set T_i, we consider the following rules. Two triplets t_i and t_j are considered potentially inconsistent if:

Rule 1
1. *have the same subject or included in the* t_i *subject* s_i;
2. *have the same object or included in the* t_i *object* o_i;
3. *have the* t_j *predicate*, p_j, *equal to not* t_i *predicate*, p_i;

Rule 2
1. *have the same subject or included in the* t_i *subject* s_i;
2. *have the same object or included the* t_i *object* o_i;
3. *have the* t_j *predicate*, p_j, *equal to the “opposite” of the* t_i *predicate*, p_i;

Rule 3
1. *express an interaction*;
2. *and have different objects*;

Rule 4
1. t_i *expresses a constraint*;
2. t_j *object is the subject of* t_i;

Rule 5
1. *have a different number of preconditions or*;
2. *every precodition of* t_i *is equivalent to at least one precondition of* t_j *and viceversa*.

Two triplets t_i and t_j define an interaction if they have:

Rule 3.1
1. *the* t_i *object is the* t_j *subject*;
2. *the* t_j *subject is the* t_i *object*.

A triplet t_i expresses a constraint or a precondition of another triplet t_k if:

Rule 4.1
1. t_k *occurs only if* t_i *occurs*.

Furthermore, two triplets t_i and t_j are considered potentially equivalent if they:

Rule 6
1. *have the same subject or included in the* t_i *subject* s_i;
2. *have the same object or included in the* t_i *object* o_i;

3. *have the same predicate, p_j, equal or included in t_i predicate, p_i, or have respectively predicates "constraint" and "precondition".*

Rules from 1 to 4 allow to verify internal consistency of software requirements triplets. Rule 5 allows to identify inconsistencies between conditioned triplets to verify both internal inconsistencies in software requirements and inconsistencies with FMEA. Rule 6 allows to verify the presence of FMEA actions among software requirements functions.

3.2 The Framework

Figure 1 shows a schema of the proposed framework implemented in Java. The green boxes represent documents, the blue boxes represent the software functions, the orange boxes represent the input rules according to which software functions check inconsistencies. Input documents are pre-processed according to some NLP basic rules. Then, they are processed by the software function Triplet Extractor. It transforms the input documents into a set of triplets reported in text files: respectively, File set *Ri* related to software requirements triplets and File set *Fi* related to FMEA triplets. Input documents must have predefined formats. FMEA file is processed by

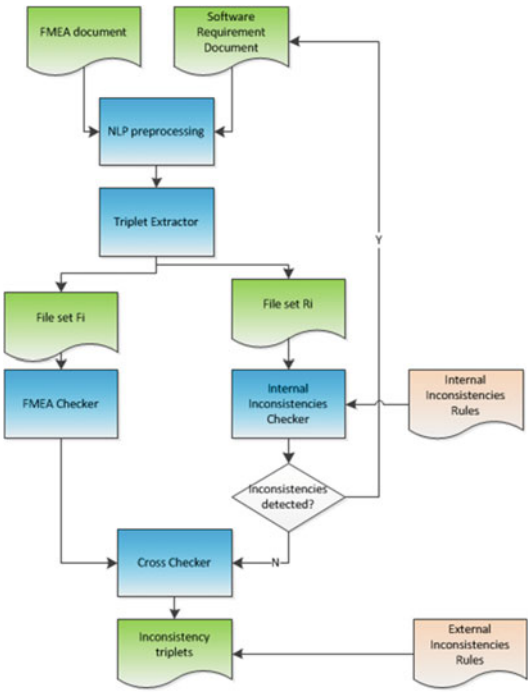

Fig. 1 The software framework

FMEA checker. It verifies the completeness of each row and groups the equivalent actions, in order to have the complete set of preconditions for it. Requirements triplets are checked for internal consistency by the Internal Consistencies Checker function according to the rules identified in Sect. 3.1. In the case of possible inconsistencies, the file is verified and manually updated, and the processing starts again until any internal inconsistency is removed. At the end, the two text files are given in input to the Cross checker function, which implements the rules related to external consistency. In the specific case study, it checks if each triplet in FMEA is equivalent at least to one triplet in the requirements set. In the negative case, it highlights a possible inconsistency. In the positive case, it checks if preconditions are equivalent to the current triplets. In the negative case, it highlights possible inconsistencies. To detect contradictory or equivalent triplets, the software function searchs the relative terms in the ontologies and evaluates semantic distances by means of Wu and Palmer semantic metric. To detect if a term x is equivalent to a term y, it identifies the set of all synonymous of y, verifies that x does not belong to it and evaluates the medium of distances between y and each synonym. If the distance between x and y is between 0 and such medium, x can be considered equivalent to y. To detect if a term x is opposite to a term y, it identifies the set of all antimony relations of y, verifies that x does not belong to it and evaluates the mean of distances between x and each word of such set. If the distance between x and y is major of the evaluated distance, x can be considered opposite to y as well as is more "semantically related" to the opposite terms.

The combination of such basic algorithms with the inconsistency rules provides the underline logic of the verification framework.

3.3 Experimental Results

In our case study, we are interested to detect inconsistencies between a part of system FMEA and a set of fifty software requirements. FMEA is relative to an unmanned space vehicle and requirements are related to its on-board data handling software. In such set, at least ten sentences express well-known inconsistencies. Requirements describe some software functions: the mission organization in different phases from pre-launch to vehicle landing; the exception reporting to the ground segment; the issuing to ground segment of actual collected data related to vehicle health status; the safety constraint, according to which ground operator shall always know the position of the vehicle; the typical reaction to an on board over temperature problem. Hereafter, the set of inconsistent software requirements related to the on board software:

R1 it shall not never power off equipments until deceleration phase;
R3 it shall disable reporting in preflight;
R4 it shall activate the flight termination system only during flight or deceleration phase in the case of RF power amplifier failure;

System				
Subsystem			Failure Mode and Effects Analysis	
Phase	Pre flight			
Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Potential Cause(s)/ Mechanism(s) of Failure	Recommended Action(s) to Software
RF power amplifier	no function	Loss of mission	Loss of physical conection between amplifier and power amplifier	
		Loss of mission	Temperature overcomes 80°	Power off the power amplifier
	Intermittent function	Degraded mission	Temperature overcomes 80°	Power off the power amplifier
GPS	No function	Mission degradation due to loss of accurate position evaluation	GPS phisical damage	Reports the problem to ground
			Phisical connection with GPS	Reports the problem to ground
Flight Data recorder	No function	Post flight analysis can't be done	Loss of physical connetion between on baord computer and FDR	
		Post flight analysis can't be done	Mass memory full	Disable on board software storing function
	Untimely function	Post flight analysis can't be accurate	on board software tasking not working	Disable on board software storing function
	erroneous function	Post flight analysis can't be done	Mass memory damaged or on board software not working properly	
bus Spacewire	No function	Loss of mission	loss of physical connetction	Activate the flight termination system
Software scheduling	Intermittent function	Degraded mission		Reboot the software

Fig. 2 Little part of FMEA with known inconsistencies

R5 it shall reboot itself if it does not receive data from equipment.

The part of FMEA under study with known inconsistencies with the above requirements is reported in Fig. 2. The algorithm processed at least 50 requirements triplets and 50 FMEA rows. Some examples of known inconsistent triplets between requirements and FMEA are reported in Fig. 3. R1 and F1, and R3 and F3 are in conflict

Requirement		FMEA Row	
R1	R1.1 <OBSW, not power_off, equipments> R1.2 <OBSW, power_off_IN, deceleration>	F1	OBSW shall power off RF power amplifier if temperature overcomes 80° F1.1 <OBSW, power-off, RF power amplifier> F1.2 <OBSW, power-off_IN, preflight> F1.3 <RF power amplifier, overcomes, 70°> F1.4 <F1.3, pre-condition, F1.1>
	Any corresponding requirement	F2	OBSW shall disable storing in the case of memory full F2.1 <OBSW, disable,storing> F2.2 < memory,is,full> F2.3 <OBSW_disable_IN, preflight> F2.4 <F1.2, pre-condition,F1.1>
R3	T11 <OBSW,disable,report> T12 <OBSW, disable_IN, preflight >	F3	OBSW shall report errors to ground in the case of GPS malfunction F3.1 <OBSW, report, error> F3.2 <OBSW, report_to, ground> F3.3 <OBSW, report_IN, preflight> F3.4 <gps, failure, true> F3.5 <F3.4, pre-condition, F3.1>
R4	T4.1 <OBSW, activate, FTS> T4.2 <OBSW, activate_IN, flight > T4.3 <OBSW, activate_IN, deceleration> T4.4 <RF power amplifier, failure, true> T4.5 <T4.4, pre-condition, T4.1>	F4	OBSW activate the flight termination system F4.1 <OBSW, activate, FTS> F4.2 <OBSW, activate_IN, preflight> F4.3 <RF power amplifier, failure, true> F4.4 <spacewire, failure, true> F4.5 <F2.4, or, F2.3> F4.6 <F2.5, pre-condition, F2.1>
R5	R5.1 <OBSW, reboot, null> R5.2 <OBSW, not receive, data> R5.3 <OBSW , not receive_FROM, equipments> R5.4 <R5.2, pre-condition, R5.1>	F5	OBSW shall reboot itself if tasks exceeds the assigned time F5.1 <OBSW, reboot, null> F5.2 <task, exceed, time> F5.3 <F5.2, pre-condition, F5.1>

Fig. 3 Found inconsistencies between FMEA rows and requirement

because the action required by FMEA is in contrast with that provided by requirements. F2 is inconsistent because the action required by FMEA is not present in requirements. R4 and F4 are in conflict because they have different number of preconditions. R5 and F5 are in conflict because they have different preconditions.

The program has detected all the known inconsistencies with 20 % of false positives. We think that this is mainly due to the fact that we do not use a precise word sense but we consider all word senses of each word. This could be avoided by running first the word sense disambiguation that outputs the index of the intended word sense for each element of the triplet. This index can allow to calculate distances among Wordnet synsets belonging to the given word sense. We do not think results can be influenced by the chosen metric. Obtained results encourage to go on verifying the framework. Feedback from a more exhaustive testing should help to understand the coverage level of inconsistencies in order to improve the RDF extractor model, which should also take into account the software procedures formalization, usually addressed by FMEA instead of single mitigation actions.

4 Conclusion and Future Work

In this paper, we use RDF to model software and safety artifacts like FMEA. If the set of triplets represents a single software artifact, the presented algorithm will check its internal consistency and the consistency with FMEA triplets according to identified rules. RDF allows to check consistency from a semantic point of view in a simple manner. We start our research from a simple case study: the verification of known inconsistencies between a set of 50 requirements written in English and FMEA rows. Results encourage to further investigate such approach. Future work shall include the refinement of the framework, and the extension of the framework itself, by automating all the steps of the analysis, as well as proposing a well-defined domain ontology modeling the PUS standard for space data handling software.

Acknowledgments This work has been partially supported by EU with the project CRYSTAL (SP1-JTI-ARTEMIS-2012-AIPP1-332830).

References

1. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: Proceedings of the 32nd annual meeting on Association for Computational Linguistics, pp. 133–138 (1998)
2. Nuseibeh, B., Russo, A.: Completeness in formal specification language design for process-control systems. In Proceedings of the 3rd Workshop on Formal Methods in Software Practice, pp. 75–87 (2000)
3. Nuseibeh, B., Easterbrook, S., Russo, A.: Leveraging Inconsistency in Software development. IEEE Comput. **33**(4), 24–29 (2000)

4. Mens, T., Van Der Straeten, R., Simmonds, J.: A Framework for Managing Consistency of Evolving UML Models. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.9786 (2005)
5. Kroha, P., Gayo, L.: Using semantic web technology in requirements specifications. ChemnitzerInformatik-Berichte CSR-08-02, ISSN 0947-5125, TU Chemnitz (2008)
6. Ficco, M., Daidone, A., Coppolino, L., Romano, L., Bondavalli, A.: An event correlation approach for fault diagnosis in SCADA infrastructures. In: Proceedings of the 13th European Workshop on Dependable Computing, May 2011, pp. 15–20 (2011)
7. Wordnet search 3.1. http://wordnet.princeton.edu/
8. Liu, X.Y., Zhou, Y.M., Zheng, R.S.: Measuring semantic similarity in WordNet. In: Proceedings of the IEEE International Conference on Machine Learning and Cybernetics, vol. 6, August 2007, pp. 3431–3435 (2007)
9. ISO, ISO 26262 Road vehicles Functional Safety, Part 1–10 (2011)
10. ECSS-E-40C, Safety Space Product Assurance, ECSS Secretariat ESA-ESTEC Re-quirements & Standards Division Noordwijk, The Netherlands, 6 March 2009
11. Zazzaro, G., Gigante, G., Zaccariello, E., Ficco, M., Di Martino, B.: Supporting development of certified aeronautical components by applying text analysis techniques. In: Proceedings of the 8th International Conference on Complex, Intelligent and Software Intensive Systems, pp. 602–607 (2014)
12. Ficco, M., Avolio, G., Battaglia, L., Manetti, V.: Hybrid simulation of distributed large-scale critical infrastructures. In: Proceedings of the International Conference on Intelligent Networking and Collaborative Systems, September 2014, pp. 616–621 (2014)
13. Höfig, A., Zeller, M., Grunske, L.: MetaFMEA-A framework for reusable FMEAs. In: Proceedings of the 4th International Symposium on Model-Based Safety and Assessment, pp. 110–122 (2014)
14. Gigante, G., Gargiulo, F., Ficco, M.: A semantic driven approach for requirements consistency verification. Intell. Distrib. Comput. VIII **570**, 427–436 (2015)
15. Gribaudo, M., Iacono, M.: An introduction to multiformalism modeling. In: Theory and Application of Multi-Formalism Modeling, pp. 314–329 (2013)