A Language for Writing System Specifications in an Aeronautical Context

Benoît Lebeaupin

A Language for Writing System Specifications in an Aeronautical Context

Benoît Lebeaupin
Laboratoire de Génie Industriel
École Centrale Paris
Grande voie des Vignes, Châtenay-Malabry, France
benoit.lebeaupin@centralesupelec.fr

***Abstract*—The ambiguity of natural language is an issue which predates requirement engineering. This issue is, in the general case, obviously unsolvable, nor actually needing a solution. However, we think that in particular contexts, it is feasible and desirable to reduce the ambiguity of free text specifications. We look at how specifications are actually handled in a company to be able to propose an useful but not too disruptive method for writing better specifications. We are currently developing and investigating how to validate this method.**

I. Introduction

Requirement engineering (RE), even if it is a relatively young field, is quite a vast subject with different subfields which can refer to various sciences such as computer science, ethnography and logic. The aim of the present work is to facilitate communication between actors who want to build complex systems which can comprise both software and physical parts, particularly in an aeronautical context, and could therefore be classified as of type 1.1 in Zave's classification [1].

For example, an aircraft manufacturer may want to buy landing gears from another company, an original equipment manufacturer (OEM). To do so, the aircraft manufacturer sends a specification document to the OEM stating what the landing gear system shall or shall not do. The OEM itself may want to decompose the landing gear into several subsystems, and buy these subsystems from other parties or build them in-house with different teams, requiring other specification documents. Complex systems can thus be incrementally divided into smaller parts which can be more easily designed by specialized teams.

A specification document is the way to tell the supplier what the buyer exactly wants and is a part of the contract engaging both parties. Obviously, it should be made the less ambiguous possible. However, it should also be readable easily by a lot of different persons, including aircraft certification authorities. Thus specifications need, at least for the near future, to be written mainly using natural language. Companies complain about the ambiguity of natural language and estimate that it is a major problem but cannot easily do without it.

This work aim to answer to the question "how to write less ambiguous specifications, which are however still understandable and writable easily, more processable by computers and more easily reusable ?" Eventually, we may wonder how we can avoid using natural language. The systems we want to specify can theoretically be anything, but practically, this work focuses on systems which include both physical (e.g. motors, brakes, hydraulic components...) and software parts and which interact mainly with other technical systems (rather than humans for example).

There are several technical challenges associated with the question in the previous paragraph: they focus on an interface between formal (what the system can or cannot do) and the informal (human communication during the requirement engineering process) and such interfaces are always complicated. We also want to be able to develop a method which is useful but which does not disturb too much existing processes and training, or it simply will not be used in the industry. Another challenge is the difficulty of evaluating the usefulness of such a method.

In the following section, we will look at what previous works in the field proposed to reduce specifications' ambiguity and why we think it is not adapted to the industry needs in this particular case. In the section 3, we introduce our contribution which focuses on the inputs and outputs of systems. The section 4 raises the complex problem of validation in the field of RE. Eventually, we conclude and look at various perspectives which could stem from our work.

II. Related Work

We should begin by stating that "requirement engineering" covers a lot of different things. For example, the "requirements" as the atomic components of an aeronautical system specification have few things in common with the high-level "goals" of Goal-Oriented Requirements Engineering (GORE) in a software engineering context. A large part of research works (such as i* [2] or KAOS [3]) in the RE domain thus concern a different part of the RE process compared to what we need for this work.

Another discrepancy is that, understandably when we look at the origin of the field, most of RE works concern software-only systems, but the systems we focus on are physical systems which may include software. What differences does it make? Firstly, software systems are usually considered as discrete and various RE methods are based on this (such as the transition axiom method [4], or Event-B [5]), but the inputs and outputs of a motor or the ambient temperature for example, are continuous and are better modeled as continuous. Another difference is

that physical systems directly interact with the physical world, which adds less predictable and less formalizable parameters.

The matter of formalization is something important in specification because it is closely linked to ambiguity: a formal language is usually non ambiguous, but is using a formal language the only option to communicate unambiguously? One obvious problem is that a formal language allows to communicate only with those who know it. Another problem is that because of complex interactions with the physical world, system engineers usually agree that a completely formalized physical system specification is impossible. Thus we did not try to develop a self-contained, completely formal language such as Event-B, but rather tried to create something less ambiguous than free text, but more expressive than a formal language.

One of the ways to make natural language less ambiguous is to restrict it, for example by using templates such as "The *system* shall *do something* with *a given level of performance* in *a given context*" [6]. It is a general advice often found in RE fundamentals books, however books' authors themselves warn against its mandatory use. In [7], the authors propose 5 templates which can be composed to support the writing of various requirements.We think that these kinds of templates are both too generic because they may not help engineers a lot and too restrictive because a lot of requirements do not fit the templates.

Controlled Natural Languages (CNL) are restrictions of natural language by various means, depending on the type of CNL. We can identify two main type of CNL:

- "Simplified" or "technical" languages such as Simplified Technical English (STE) [8] are made to facilitate human communication, particularly for non-native speakers, and are used, for example, for technical documentation.
- "Logic-based" languages such as Attempto Controlled English (ACE) [9] aim to facilitate human-computer communication

Both types of languages have usually a controlled vocabulary, composed of predefined words (which are close to protected words in programming languages) and content words, defined by the user. These languages also restrict the syntax, but using different rules, adapted to the language purpose: For ACE, we can only write sentences which are automatically and unambiguously translatable into first-order logic, easily "understandable" by a computer, whereas for "technical" languages, we could have rules such as "sentences will be composed of less than 15 words", since long sentences are harder to understand for humans. Our work is closely related to "logic-based" languages: we essentially search to have a language of this type which includes the needs of requirement engineers.

The authors of [10] automatically extract models from textual requirements and check that these models satisfy certain properties. This "lightweight formal method" is interesting and relatively economical, however it seems to need requirements which use a very precise language and are highly structured and regular. The aim of our work is to be able to write this kind of requirement with the less additional cost possible.

Our work builds upon various sources[1], among these, an interesting article is [11], which will be mentioned several times in the next section. We however did not find a lot of works focusing on the precise needs presented here, even if there is a real industrial demand for such methods and tools. The CRYSTAL [12] project is focusing on this aspect: ontologies and patterns are used to obtain less ambiguous and more processable requirements [13]. We believe our work is different and complete these efforts by proposing a more flexible approach than static, predefined patterns and by requiring less effort than the construction of a complete ontology before beginning to write a specification.

III. Considerations on Specification Writing

A. Observations and First Goals

Together with a literature search, we began our work by analyzing specification documents sent to or written at Safran and by discussing with the persons responsible of requirement engineering in the company. The following observations are thus obviously not representative of the state of the art of academic RE, but we think they represent reasonably well what actually happen in the industrial context presented earlier.

- It is practically impossible to write a non-trivial, completely formal specification.
- It is however possible to rewrite at least some parts of existing specifications to make them less ambiguous.
- This kind of specifications is created using preexisting work, such as client specifications and engineering studies. The usual step of requirement elicitation is largely absent, at least at the OEM level.
- We often find the same, or almost the same, requirements in different specification documents.
- There are confusions between requirements, assumptions, and "hypotheses", where assumptions, in the sense of [11], "describe the environment as it is in the absence of the machine or regardless of the actions of the machine" and hypotheses are implementation biases: suppositions the customer makes about the design of the system.

We think that writing "better" requirements will allow us not only to have more readable individual requirements, but will also make the complete process easier (for example, to determine if some given requirements should be linked to other lower- or higher-level requirements).

How can we write "better" requirements? The first axis we identified is to write two equivalent things the same way. It applies of course to the vocabulary (for example, if the studied system is a landing gear, it should not be called "the system" in a requirement and "the landing gear" in another), but also to syntax[2]:

- "`The system shall transition from OFF to STANDBY state when all the`

[1] including internal methodology at Safran

[2] We will see later that the "states" mentioned here are not a good practice in RE, but it does not change the pertinence of the example

`following conditions are met: -1) The signal ON is true -2)...`"[3]

and

- "`The system shall transition into STANDBY state when all the following conditions are met: -1) the OFF state is true -2) The signal ON is true -3)...`"

are strictly equivalent, but written differently.

How then can we determine if two things are equivalent? Basically, if they refer to the same "concept". This is related to what Zave and Jackson call a "designation" in [11], an informal explanation of a term. Additionally, since we agree with their statement that "a specification should contain nothing but information about the environment", what kind of concepts can be mentioned in a specification? Zave and Jackson classify "actions"[4] into three different types:

- environment-controlled and unobservable by the system,
- environment-controlled and observable by the system,
- system-controlled and observable by the system.

They consider that actions cannot be controlled by the system if it cannot observe them. Additionally, we think that unobservable actions are not relevant to specifications: if we need to mention something in a specification, then it must have an impact on the system. If this impact can be unambiguously traced back to observable actions controlled by the environment, it should be done to clarify the specification instead of having unobservable actions. If this impact cannot be traced back to observable actions controlled by the environment, then it means either there is no impact, or we do not know it and this will cause ambiguity.

We therefore want to restrict what is present in a specification to the part of environment which directly interacts with the system. The distinction inputs/outputs (environment-controlled/system-controlled) is less absolute for physical systems than it may be for software[5], but it is still a practical way to describe things, so we will use these terms in the rest of the paper.

B. A Proposition of Formalization

Concretely, how can we use what is presented earlier to write less ambiguous specifications? We consider that each requirement is a boolean formula, basically: "if we have such properties on the inputs, we want such properties on the outputs". The system respects the specification if all requirements are true. Attached to each input/output (I/O) would be values, such as false/true for discrete signals, or rotational speed, torque or current for physical I/O. Using, for example, relational operators, we can write atomic formulas and compose them using boolean operators to create a requirement. For example, the requirement "`if the signal CMD_FWD is true and the rotational speed of the wheel axis is less than 18 rpm then the system shall generate a 5 N.m torque on the wheel axis`" can be decomposed into $(CMD\_FWD = 1)$, $(rot\_speed_{wa} < 18$ rpm$)$ and $(T_{wa} = 5$ N.m$)$, and connected using a boolean "and" and a boolean "implies", where rot_speed_{wa} is the input modeling the rotational speed of the wheel axis and T_{wa} the output modeling the torque on the wheel axis.

The textual requirement is more readable and writable, especially for non-technical people, than the "logical" form $((CMD\_FWD = 1) \wedge (rot\_speed_{wa} < 18$ rpm$)) \Rightarrow (T_{wa} = 5$ N.m$)$, which explains why we are still using natural language for specification documents. Therefore requirements should, at least for now, stay in textual form. However, we think that encouraging requirement engineers to write textual requirements which can be translated to a logical substrate should reduce requirement ambiguity. An interesting question is which logical substrate among the various existing ones do we choose for our language?

We have analyzed several different industrial specifications as well as talked with requirement engineers, and continue to do so, to be able to know what constructs are typically needed in a specification. There are two necessary kinds of constructs: construct to get atomic, boolean-valued formulas from I/O values (e.g. relational operators) and operators to get formulas from other formulas (e.g. boolean "and"). For the moment being, we have identified classical constructs ("and", "or") or more domain-specific ones, such as "the probability of a failure leading to X shall be less than p", where X is a boolean formula (for example "`the rotational speed of the wheel axis is null`" which could be written "$rot\_speed_{wa} = 0$") and p a probability, usually expressed in events per flight hour in an aeronautical context.

We have to be cautious when adding such constructs to our language since we do not want to reintroduce the ambiguity we aim to remove by using our language, but these construct may not be formal: in the previous example, "failure" or "leading to" cannot be formally defined. Since these expressions are used in actual specifications and engineers do not seem to have any problem interpreting them, we have to assume that these expressions are not ambiguous even if they are not formally defined, and are what Chantree and al. would call "innocuous ambiguities" in [14]. We have to find, and integrate in the language, the constructs which are useful for specification writing and either formally defined or not formally defined, but understood the same way by all potential readers and writers of a specification. It may seem to be an impossible task, but we should note that all current system specifications are made using natural languages and the assumption that everyone understand the same thing when reading the same sentence. Our aim is not to create perfectly formal specifications, but to write "less ambiguous" specifications.

We wrote earlier that environmental variables which are not identified as I/O of the system should not appear in the

[3]Sentences written in `this font` are requirements or parts of requirements, usually taken from an industrial specification document. They were modified to make them generic for obvious intellectual property issues

[4]Once again, this is a software, event-based point of view, in our context, "action" could be replaced roughly by "property of the environment"

[5]For example, the rotational speed of a motor axis may be an output for a functional requirement and an input for a safety requirement

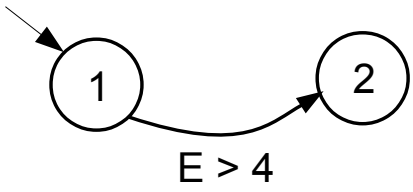

Fig. 1. A simple state machine, "E > 4" is a guard

specification, but an user may want to write a requirement such as "`during takeoff, the system shall...`" and "takeoff" is not an input of the system. Firstly, as we proposed, the user should be able to decompose "takeoff" into one or more properties on one or more inputs of the system. If he cannot do that, how is the system supposed to "know" that the airplane is taking off? Assuming the user does have such a decomposition, one problem could be that this decomposition is a complicated formula A, which would reduce the clarity of the specification and could prevent the engineers from understanding what is the "physical" meaning of a requirement where A appears. One conceptually simple way to solve this problem would be to define the string "takeoff" as equivalent to the formula, the same way we create new variables which refer to the results of complicated formulas in a program, and most importantly, to precise that you're doing so in (the preamble of) the specification. Thus the original requirement would stay the same, but "takeoff" would refer to a formal definition. "takeoff" is now what we call a "composed input".

C. The Issue of "States"

When we look at real specifications, we notice that the example given in the previous subsection is quite simple: almost no requirement are simply combinational, and they usually refer to "states" (for example "`In state BKWD, if the rotational speed of the wheel axis is less than 18 rpm and the signal CMD_BKWD is true, then the system shall generate a -5 N.m torque on the wheel axis`"). These states are sometimes defined in the specification by cryptic requirements such as "`The system shall have an ON state`", but specifications never define exactly what is a state, nor how a system can "have" them. It can create quite ambiguous requirements such as "`The probability of inadvertent activation of the system (transition from OFF to STANDBY or ON state) shall be less than` 10^{-8} `per Flight Hour`".

This problem, mentioned for example in [11], is mainly caused by the fact that such requirements are a blatant implementation bias. If we consider, as Zave and Jackson do, that requirements can only be statements about the environment, the states exist only in the environment. We consider that states are simply what we introduced as "composed inputs" in the previous subsection, which are simple names used as placeholders for properties on the environment of a system. A simple example could be that the condition "`when in state 2`", where the state 2 is defined by the state machine given in fig. 1, is equivalent to the condition "`if the value of the input E has been greater than 4 in the past`". Of course, the example is quite simple, but the principle does not change for a more complicated state machine: the so-called "states" are just a faster way to write properties, particularly temporal properties, on the environment of a system. As for other composed inputs, the definitions of the states should be explicitly written and identified as such in the specification.

D. More on Inputs/Outputs

What are exactly the inputs and outputs that we mention in this paper? First of all they are abstractions of properties of the environment. The difficult question is how much abstracted should an I/O be? For example, a three-phase line power supply could be modeled as one input with a "power" value, or 2 inputs of value "tension" and "current", or 6 inputs, 2 tension-current couple for each phase. The optimal abstraction level will obviously depend on how the I/O is used in the specification. A simple abstraction is easier to model but will not have the same expressing power as more complicated ones. For example, if we model a 3-phase line with one "power" input, it will not be possible to express a phase loss, but in the 6-input abstraction, a phase loss is expressed simply by having a null current for one of the phase.

One of the way we could "give back" this expressing power to the simpler abstractions would be to add "attributes" to I/O, for example the 3-phase line could be modeled as one input with a "power" value and two attributes "electrical" and "three-phased". If we know that "three-phased" "electrical" inputs can lose a phase, and have written it in an ontology for example, we can then write requirements such as "`If (the power supply) have (a phase loss), the system shall...`". As for what we wrote earlier, an attribute and its properties may not be formally defined if we consider that everyone have the same interpretation of it.

We mentioned that what is written in a specification, like I/O, must refer to some concepts so we can know what they mean. We can wonder if the cost of defining these concepts is too important compared to the eventual profits, and would make what we are proposing useless. However, in the specification documents we study, most inputs and outputs (I/O) are defined, although usually incompletely and imperfectly, to give a context for the system specification or simply to help define the system boundaries (for example by drawing a box representing the system, linked with arrows to other boxes representing other systems in the environment). So even if this kind of information is not, as far as we know, currently used how we envision it, companies do dedicate resources to create it and write it. It means our proposal requires less resources and adaptation:

one of the point of our work is that the I/O we use should correspond relatively well to the I/O used by system engineers when they are writing "boxes and arrows" architectural models such as SADT, so we can reuse what was done in these models to write better specifications.

E. How Can We Treat Non-Formalizable Requirements?

The principle we present here is quite generic, and we think that all requirements could be written in the form "if we have such properties on the inputs, then we want such properties on the outputs". However, we do not think it is a good idea, because the effort needed for the conceptualization and formalization of various inputs and outputs is too important. For example, a requirement may demand that "`The materials used in the system shall not suffer corrosion under the environmental conditions described in document A Category C`", where document A is an external, possibly normative document. We think this is an useful requirement and that formalizing, in the specification, the "inputs" it refers to (for example, the levels of humidity, pH, percentage of various chemicals in the ambient air...) is possible, but it would require too many efforts for almost no profit, since it is presumably already done in document A and it would reduce the readability of the specification. We notice the same thing for a lot of requirements: they are not "formal" and it would be too difficult to model the I/O they implicitly refers to, so we have to continue using free text and try to make them as unambiguous as possible. Thus, when writing a specification document, there should be a compromise to find: at which point do we stop modeling I/O and write the usual free text requirements?

We think this compromise is more interesting in the aeronautical context, since the systems we focus on are technical systems, mostly surrounded by other technical systems, whose I/O are relatively easy to represent and formalize. It is not a surprise that the majority of the "difficult-to-formalize" requirements are those which involve humans, such as requirements related to maintenance.

One way to reduce the effort needed to write these requirements is that some of them are what we could call "generic" requirements: requirements which are present in every aeronautical system specification. For example, "`All units and sub-units shall be protected against Electro Static Damage (ESD) during repair, exchange, maintenance and handling`" could be found in any specification. For companies which are designing several systems for the same context, it may be useful to have a library of well written requirements of this type, which is automatically added to a new specification document, in a crude yet simple reuse strategy.

Related to these "generic" requirements, another type could be the "parametric" type: requirements that change only by a numerical value (for example "`The service life target of the system shall be 24,000 flight cycles`") or only by the document or part of the document it refers to, like in "`system shall not suffer corrosion under the environmental conditions described in document A Category C`". Once we know the pertinent information (for environmental requirements such as the second example, the exact reference is typically given by the location of the system in the aircraft) engineers could directly generate this kind of requirements without having to actually write them. Once again, this is a relatively crude method for reusing requirements and it was not tested to see if it is profitable, but it could be useful and it is not necessarily done in the industry.

TABLE I
CLASSIFICATION OF AN INDUSTRIAL SPECIFICATION

	Number of requirements in the category	Percentage of requirements
I/O formalism	67	39.2
generic	25	14.6
parametric	14	8.2
rest	65	38.0
total	171	100

F. Preliminary Results

To test the principles we propose, we tried to put each requirement from a real aeronautical system specification from Safran into one of these categories "generic", "parametric" (both introduced in the previous subsection) or "translatable into the I/O formalism", the results are given in the table I. Some remarks on these results:

In the first place, the classification of the various requirements in these categories was not an exact process, for several reasons: we considered that a requirement was translatable in our formalism if we could find a translation and that translation required only a "reasonable" effort for modeling the inputs and outputs. Moreover, as all translation, this translation is rarely exactly equivalent to the original requirement, especially since the aim of this work is to better write requirements and that some requirements in this specification were quite ambiguous. Since we did not have access to all the specifications of the company, we did not know for sure that "generic" and "parametric" requirements are used in all the specifications.

These results are here only for illustration and we do not claim that it proves that our work is useful, but our industrial partners think they are interesting and that such cover rates are sufficient to be of economic interest. The specification was neither a completely finalized work nor a rough first draft, and this is interesting because it was a relatively good specification, but we could still find errors or ambiguity.

The requirements we could not put into the previous categories because they were both too specific to the studied system and too difficult to formalize may still be dealt with using another method. For example, some were envelope requirements: they defined where the system could be installed in the airplane. We could imagine that these requirements could

be replaced by one requirement referring to an external CAD model describing the installation envelop of the system.

IV. Evaluation of RE Practices

As mentioned in [15], few works in the RE field concern or even mention the scientific evaluation of RE practices. It is an important problem since if we don't have a serious evaluation of the method presented in a paper, we, and industrial actors, cannot be sure that this method is really useful. Experimentation is, after all, the basis of scientific method. The main issue is how to do these evaluations: the evaluation of a complicated design process involving humans and subtle communications between them, or even of a part of this process, is something quite complicated, especially since RE authors usually come from a computer science background and may not know experimental methods from, for example, social sciences, which may be quite adapted to the difficulties of evaluation in RE.

As a beginning researcher in the field of RE, the author of this paper would like to be able to know if the ideas presented here are useful or how they can be improved and wonders if there exist standard, "benchmark" tests, which could be used to compare the different steps of various RE methods.

V. Conclusion

We searched how aeronautical system specifications were written at Safran and how they could be improved, notably by discussing with engineers responsible of RE in the company. We believe that it is possible to make requirements less ambiguous and thus to improve the whole process of RE. One of the goal of this work is to propose a method which could realistically be adopted in the industry: specifications would be less ambiguous if they were (and could be) written in first order logic instead of natural language, but it is simply not possible to ask a whole industry to learn first order logic.

We searched in the fields of RE, language generation, natural language processing and controlled languages for methods which could be used to reduce the ambiguity of requirements in specifications. We did not find methods which could be directly applied to our problem, but some ideas and general concepts were found to be useful and constitute the foundation of our work.

We proposed principles aiming to reduce the ambiguity of natural language system specifications while trying to keep most of the expressiveness and ease of use of natural languages. Our work is designed to reuse as much as possible the efforts already done in a company when it needs information. The core of our work consists in writing the requirements as boolean formulas, where atomic formulas are properties on the environment (the inputs and outputs of the specified system). The results of this process are not formal requirements, but we are trying to obtain requirements which are the less ambiguous possible, by linking the concepts used in a requirement to exterior and agreed upon definitions. Of course, the work on both theoretical and practical aspects of our proposal is not finished and will be continued in the following years as the author finish his PhD.

We think it is impossible to write a complete specification without unconstrained natural language and some part of ambiguity, so any specification writing method should be able to tolerate free text, and compromises have to be found by requirement engineers between an easy to write, but potentially ambiguous or even wrong, free text and harder to write, but hopefully less ambiguous, constrained language.

Among future works, beside the needed development and scientific evaluation of our ideas, we think that we could write at least some parts of specifications using more or less formal models instead of natural language, and if we ever need to have these parts written in natural languages, the models could be automatically translated into natural language requirements. Another perspective would be to be able to realize automated reasoning on the "formalized" parts of the specification.

Acknowledgment

The author is supported by the Blériot-Fabre chair, co-directed between CentraleSupelec and Safran.

References

[1] P. Zave, "Classification of research efforts in requirements engineering," *ACM Computing Surveys (CSUR)*, vol. 29, no. 4, pp. 315–321, 1997.

[2] E. Yu, "Modelling strategic relationships for process reengineering," *Social Modeling for Requirements Engineering*, vol. 11, p. 2011, 2011.

[3] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Fifth IEEE International Symposium on Requirements Engineering*. IEEE, 2001, pp. 249–262.

[4] L. Lamport, "A simple approach to specifying concurrent systems," *Communications of the ACM*, vol. 32, no. 1, pp. 32–45, 1989.

[5] J.-R. Abrial, *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.

[6] K. Pohl, *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[7] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (EARS)," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*. IEEE, 2009, pp. 317–322.

[8] ASD Simplified Technical English. [Online]. Available: http://www.asd-ste100.org/

[9] N. E. Fuchs and R. Schwitter, "Attempto Controlled English (ACE)," in *Proceedings of the First International Workshop on Controlled Language Applications*, 1996.

[10] V. Gervasi and B. Nuseibeh, "Lightweight validation of natural language requirements," *Software: Practice and Experience*, vol. 32, no. 2, pp. 113–133, 2002.

[11] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 1, pp. 1–30, 1997.

[12] CRitical sYSTem engineering AcceLeration (CRYSTAL EU Project). [Online]. Available: http://www.crystal-artemis.eu/

[13] A. Fraga, J. Llorens, L. Alonso, and J. M. Fuentes, "Ontology-assisted systems engineering process with focus in the requirements engineering process," in *Complex Systems Design & Management*. Springer, 2015, pp. 149–161.

[14] F. Chantree, B. Nuseibeh, A. De Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in *Requirements Engineering, 14th IEEE International Conference*. IEEE, 2006, pp. 59–68.

[15] B. H. Cheng and J. M. Atlee, "Research directions in requirements engineering," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 285–303.