Language and communication problems in formalization: a natural language approach

Alessandro Fantechi[1,2], Stefania Gnesi[2], and Laura Semini[2,3]

[1] Dipartimento di Ingegneria dell'Informazione, Università di Firenze
alessandro.fantechi@unifi.it
[2] Istituto di Scienza e Tecnologie dell'Informazione "A.Faedo",
Consiglio Nazionale delle Ricerche, ISTI–CNR, Pisa
stefania.gnesi@isti.cnr.it
[3] Dipartimento di Informatica, Università di Pisa
laura.semini@unipi.it

Abstract. "The bride is dressed in red and the groom in white." Sometimes someone cannot believe their own ears, thinking they have misunderstood, and instead the communication is clear and exact, Egon actually got married in white and Donatella was in red. Some other times someone believe they have understood and instead the message is ambiguous and unclear.
When considering software requirements, one way to eliminate inaccuracies is to build a ground model and give it a formalization. In this paper, we propose an approach that begins by searching for terms and constructs that may cause communication problems and suggests a systematic way to disambiguate them.

1 Introduction

In the initial step of the classic software development process requirements are defined. This process aims to obtain a general description of functional (what the program should do) and non-functional (such as resources, performance, etc.) requirements. Although this process is somewhat systematic, requirements identification is usually intuitive and informal, and requirements are usually expressed informally through natural language phrases. Requirements are typically the basis on which implementations are built. However, natural language is inherently ambiguous, and ambiguity is seen as a possible (if not certain) source of problems in the subsequent interpretation of requirements.

Among the others, Egon Boerger advocated to start from an unambiguous, formal, *ground* model and proceed through a rigorous formal process that preserves correctness to develop the implementation [4, 5]. Defining ground models is one of the three activities that constitute the formal method for engineering computer-based systems known as Abstract State Machine (ASM) method [6].

However, this introduces a necessary formalization step that should bridge the informal understanding of what is expected from the system to be built into its formal, rigorous model. Quoting Egon:

> The notoriously difficult and error prone elicitation of requirements is largely a formalization task in the sense of an accurate task formulation, namely to realize the transition from usually natural-language problem descriptions to a sufficiently precise, unambiguous, consistent, complete and minimal formulation of ... "the conceptual construct" ... of a computer-based system, as distinguished from a software representation, e.g. by code.
>
> This is mainly a language and communication problem between the domain expert or customer and the software designer who prior to coding have to come to a common understanding of "what to build", to be documented in a contract containing a model which can be inspected by the involved parties.

This, together with

> the fact that there are no mathematical means to prove the correctness of the passage from an informal to a precise description

makes the formalization step quite challenging, especially for complex systems. Egon's approach has been that of expressing "the conceptual construct" as a so called "ground" model:

> ground models must be apt to mediate between the application domain, where the task originates which is to be accomplished by the system to be built, and the world of models, where the relevant piece of reality has to be represented.

Our research has instead focused on the informal nature of requirement themselves. Pioneering a popular stream of research in the Requirements Engineering discipline, we have addressed the said communication mismatch problem by focusing on the identification and analysis of *ambiguity* in requirements documents. In particular, in previous works we have focused on the automated analysis of requirements documents by means of Natural Language Processing (NLP) tools [7]. This kind of analysis is aimed at identifying typical natural language defects, especially focusing on ambiguity sources [3].

In this paper we exemplify the principles of this research stream by applying the QuARS NLP tool to identify ambiguities in the informal description of the known Steam Boiler case study [2], showing how also that natural language description (considered as if it were a requirements document for the proposed case study) suffered from ambiguity: this ambiguity has necessarily to be solved when formalising the case study (and we show examples of solved ambiguities).

2 Sources of ambiguity in requirements

Natural language (NL) is the most commonly used means of expressing software requirements despite the fact that it is inherently ambiguous, and despite the fact

that ambiguity is a known source of problems in the subsequent interpretation of requirements.

Natural language processing techniques (NLP) are frequently used today to help system engineers and project managers analyze and correct requirements in natural language and to provide the user with a quality assessment of each requirement analyzed, significantly automating and speeding up the task of finding possible errors in NL requirements documents. However this does not mean that human review of the requirements is unnecessary or unimportant. Rather, these NLP techniques and tools will provide a help in this task. Domain experts shall review the results the tools provide and use their expertise to correct the deficiencies they find.

There are many different sources of ambiguity that can cause communication problems between stakeholders, some are due to the use of ambiguous words (*lexical* ambiguities), others (the *syntactical* ambiguities) to particular syntactical structures or to the position of a word in a requirement (when used as an adjective or pronoun for example). We list the most common sources of ambiguity, with a classification inspired by [3].

Analytical, *attachment*, and *coordination* ambiguities, are different forms of syntactic ambiguities. They occur when a sentence admits more than one grammatical structure, and different structures have different meanings. We provide some examples, for a more in-depth discussion we refer to [3]: "software fault tolerance", can be tolerance to software faults or fault tolerance achieved by software (*analytical*); "I go to visit a friend with a red car" means either that I go with the red car or that the friend I'm visiting has a red car (*attachment*); "She likes red cars and bikes", where it is not clear whether she likes all bikes or only red ones (two *coordinators* in a sentence); "I will travel by plane or ferry and car", that can be interpreted as "I will travel by plane or I will travel by ferry and car" or as "I will travel by plane or ferry and by car" (another example of *coordination*: a coordinator related to a modifier). Somehow all three have to do with associativity.

A requirement contains an *anaphoric* ambiguity when an element of a sentence depends for its reference on another, antecedent, element and it is not clear to which antecedent it refers, as in "The procedure shall convert the 24 bit image to an 8 bit image, then display it in a dynamic window" [12]. Anaphoras are another category of syntactic ambiguity and in most cases can be identified looking for pronouns.

Vagueness is a lexical defect and it is sometimes considered not as an ambiguity but as a separate category, in fact it does not lead to different interpretations, but to a loose interpretation. For completeness and uniformity we keep it in this list and treat it as an ambiguity. An example is "The system must support many users at the same time": what is meant by many? 5? 1000?, without a clarification you will never be able to say if the system will have satisfied the requirement.

Temporal ambiguities are special cases of vagueness, that occur when a temporal quantification is missing. Examples are: "A message shall be sent after

activation of the registration function". How long after? "The Project Team must demonstrate mockups of UI changes to project stakeholders early in the development process". How early?

Comparatives and *superlatives* are syntactic ambiguities: a comparator without a term of comparison; a superlative with a vague or no context. Although, also in this case we can identify terms that play the role of indicators. Examples are: "The feature X may be used to provide a better grade of service to certain users". "Any emissions radiating into the driver's cab and other on-board equipment from the exterior aerial shall meet the industry standards to the highest possible degree".

We have seen that *coordinators* can lead to ambiguities in some syntactic constructions, *disjunctions* instead are a separate case because they are inherently ambiguous, as in "The user information contained in the confirmation message shall be the engine number or train number (if registered)".

An *Escape clause* contains a condition or a possibility and makes a requirement weaker, vague and unverifiable. An example is "The system, if possible, updates the data every 2 hours".

Similarly, the use of a weak verb (e.g. may or may not) leads to a *weakness* and offers an excuse for not implementing the requirement.

Quantifiers are a notable cause of ambiguity, for different orders of reasons: they can be originated from generalizations of the matter; the analysts may not have defined the universe of quantification; the analysts may not have defined the scope of the quantifier itself. The sentence "All pilots have a red car" contains two ambiguities: universe of discourse, the sentence is true in a Ferrari F1 box, not in general, and scope, the pilots share the same car?

Underspecification is a syntactical ambiguity that occurs when a generic term is used that lacks of a specifier. For instance in "It shall be possible for the driver to store numbers in the radio", it must be made clear what the numbers are. In this case the context of the requirement makes clear that they were train identification numbers.

When using a *passive voice* it is possible to omit the agent, in fact, often we change an active form to passive so that we don't have to decide on the subject of the action (which becomes the agent in the passive form). This generates a (syntactic) ambiguity that must be resolved by deciding who is the subject of the action. As an example "The fire alarm must be issued" will become "The system must issue the fire alarm".

2.1 QuARS

QuARS - Quality Analyzer for Requirement Specifications – is a tool for analyzing NL requirements in a systematic and automatic way by means of NLP techniques with a focus on ambiguity detection [9, 10].

QuARS performs an automatic linguistic analysis of a requirements document in plain text format, based on a given quality model. Its output indicates the defective requirements and highlights the words that reveal the defect.

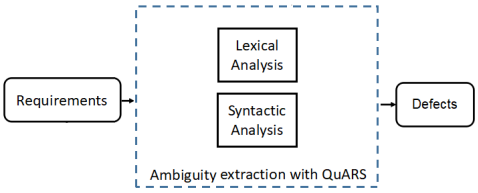

Fig. 1. The QuARS architecture for defect identification.

The defect identification process is divided into two, independent, parts (Figure 1). The first part, *lexical analysis*, detects candidate defective terms using a set of dictionaries. Lexical analysis permits to capture *disjunction, optionality, subjectivity, vagueness, optionality, weakness, quantifiers, and temporal* defects. The second part is *syntactical analysis*, that captures *implicity and underspecification* defects.

Other features of QuARS are (i) metrics derivation for evaluating the quality of NL requirements; (ii) the capability to modify existing dictionaries, and to add new dictionaries for new indicators; (iii) view derivation, to identify and collect together those requirements belonging to given functional and non functional characteristics, by defining specific requirements.

In Table 1 we present the indicators used by QuARS to detect defects of lexical and syntactic ambiguity in NL sentences. Table 2 shows how they correspond to the ambiguity classes listed above.

Ambiguity Class	**Indicators**
vagueness	clear, easy, strong, good, bad, adequate...
subjectivity	similar, have in mind, take into account,...
disjunction	or, and/or
optionality	or, and/or, possibly, eventually, case, if possible, if appropriate...
temporal	before, after, as soon as, then...
weakness	can, could, may, ...
implicity	demonstrative adjectives or pronouns
underspecification	wordings missing a qualification (e.g.: interface or information without a qualifier, such as user and price, respectively)

Table 1. Defects detected by QuARS

Ambiguity classes	*QuARS detection classes*
Analytical	n.a.
Attachment	n.a.
Coordination	n.a.
Anaphoric	n.a.
Vagueness	vagueness
Temporal	temporal
Comparatives	n.a.
Superlatives	n.a.
Disjunction	disjunction
Escape Clause	optionality , weakness
Quantifiers	quantifiers
Underspecification	underspecification
Passive voice	n.a.

Table 2. Ambiguity classes detected by QuARS

3 Steam boiler and QuARS

Based on the insight that thorough comparison between the various design formalisms proposed in the literature as well as some amount of unification would be crucial both to industrial takeover and to scientific progress in the field, an international seminar on "Methods for Semantics and Specification" was held at Schloß Dagstuhl, Wadern, Germany, on June 5-9, 1995. The seminar took the form of a "competition" between different researchers who had been invited as representatives of their particular methods. The competition was on the *Steam Boiler control* specification problem drafted by J.-R. Abrial, E. Börger, and H. Langmaack, which has been posed to the participants as a common case study [1, 2].

We consider here the natural language specification of the Steam Boiler case study as if it was the requirements document specifying the system, structuring the text as a set of numbered sentences. The resulting document is made of 39 requirements (see the Appendix). We have applied QuARS to analyse this document looking for potential ambiguities, then the output of the tool has been manually examined to distinguish true ambiguities from false positives (FP). Table 3 shows a summary of the true ambiguity defects.

The *vagueness* analysis revealed 4 defects, 3 of which are true ambiguities and only one FP. The table shows req.29, which is ambiguous because *satisfactory* is not an accurate measure for water level, and req.39 is ambiguous because it is unclear what the *appropriate* actions are. Here the communication problems

can be solved by providing ranges so that it is possible to evaluate predicates $satisfactory(water\_level)$ and $appropriate(action)$.

The *quantifiers* analysis revealed 7 defects, one of which is a true ambiguity and 6 are FP. The table shows Req.1, in which it is not clear what "*each* physical unit is connected to a control unit" means.

The *disjunction* analysis revealed 9 defects, all occurrences of *or*: 1 truly ambiguous and 8 FP. Req. 33, containing the ambiguous occurrence, introduces a case of non-determinism.

The *weakness* analysis revealed 8 defects: 2 true ambiguities (both occurrences of *can*) and 6 FP: in Req. 13 it is not clear if the program must send the program-ready signal or can avoid doing so. Req. 17 is ambiguous about the program's obligation to adjust water levels.

The search for *temporal* ambiguities returned 10 defects, out of which 9 true ambiguities and 1 FP. Most ambiguities are due, as in req. 8, to the presence of wording *as soon as*. The analyst must decide whether *as soon as* means the next step or within a some time. Similarly, requirements 11 and 37 must be clarified because of the terms *then* and *after*.

The *optionality* analysis did not reveal any defect, while *implicity* and *underspecification* revealed some defects that were false positives.

4 Solving communication mismatch problems

QuARS helps to find in the natural language specification critical points that can create communication problems as they can lead to multiple interpretations. Depending on the defect class, the range of possible interpretations changes. We address the output of the QuARS analysis of the steam boiler and indicate, for each requirement, the possible interpretations and the corresponding formalization, that depend on the class of defect. This way we provide, through examples, a systematic means to consider, for each ambiguity class as defined in Table 2, the possible interpretations and their formalization, so that they can be compared on a mathematical ground. The formalization makes use of first order logic and temporal logics, which are well suited to express in a rigorous manner the particular aspects to which a single requirement refers.

4.1 Formalising Vagueness and quantifiers

A vague term may represent an abstraction, as *satisfactory* in the example below, which is made concrete by a set of possible instances.

Req 29 "The rescue mode is the mode in which the program tries to maintain a satisfactory water level despite of the failure of the water level measuring unit"

We need to clarify what *satisfactory* means, providing a measure and a range of values. For example, let us assume it is between N1 and N2, included.

————- QuARS [Lexical] vagueness ANALYSIS ————-
The line number:
29. the rescue mode is the mode in which the program tries to maintain a satisfactory water level despite of the failure of the water level measuring unit.
is defective because it contains the wording: satisfactory
The line number:
39. once the program has reached the emergency stop mode, the physical environment is then responsible to take appropriate actions, and the program stops.
is defective because it contains the wording: appropriate
—- QuARS [Lexical] vagueness Statistics (on "boiler.txt" file): ————
Number of evaluated sentences: 39 Number of defective sentences: 4

————- QuARS [Lexical] Quantifiers ANALYSIS ————-
The line number:
1. the program communicates with the physical units through messages which are transmitted over a number of dedicated lines connecting each physical unit with the control unit.
In first approximation, the time for transmission can be neglected.
is defective because it contains the wording: each
—- QuARS [Lexical] Quantifiers Statistics (on "boiler.txt" file): ————
Number of evaluated sentences: 39 Number of defective sentences: 7

————- QuARS [Lexical] Disjunction ANALYSIS ————-
The line number:
33. as soon as the water measuring unit is repaired, the program returns into mode degraded or into mode normal.
is defective because it contains the wording: or
—- QuARS [Lexical] Disjunction Statistics (on "boiler.txt" file): ————
Number of evaluated sentences: 39 Number of defective sentences: 9

————- QuARS [Lexical] weakness ANALYSIS ————-
The line number:
13. as soon as a level of water between N1 and N2 has been reached the program can send continuously the signal program-ready to the physical units until it receives the signal physical_units_ready which must necessarily be emitted by the physical units.
is defective because it contains the wording: can
The line number:
17. as soon as the water level is below N1 or above N2 the level can be adjusted by the program by switching the pumps on or off.
is defective because it contains the wording: can
—- QuARS [Lexical] weakness Statistics (on "boiler.txt" file): ————
Number of evaluated sentences: 39 Number of defective sentences: 8

————- QuARS [Lexical] temporal ANALYSIS ————-
The line number:
8. as soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero.
is defective because it contains the wording: as soon as
The line number:
11. if the quantity of water in the steam-boiler is below w then the program activates a pump to fill the steam-boiler.
is defective because it contains the wording: then
The line number:
37. this mode can also be reached after detection of an erroneous transmission between the program and the physical units.
is defective because it contains the wording: after
—- QuARS [Lexical] temporal Statistics (on "boiler.txt" file): ————
Number of evaluated sentences:39 Number of defective sentences:10

Table 3. Analysis with QuARS of the Steam Boiler Requirements

If we consider as a satisfying range the closed interval [N1..N2], using first order logic we may write:

$$\forall x.\ (satisfactory\_water\_level(x) \leftrightarrow (x \geq N1 \wedge x \leq N2))$$

and then formalise Req 29 as follows:

$$rescue\_mode \rightarrow \exists x.(water\_level(x) \wedge satisfactory\_water\_level(x))$$

Quantifiers can make a requirement ambiguous for two reasons at least: the universe of discourse and scope.

Req 1 "The program communicates with the physical units through messages which are transmitted over a number of dedicated lines connecting each physical unit with the control unit. in first approximation, the time for transmission can be neglected."

Here the universe for *each* is clear, and it is the set of physical units. The problem is the scope and the two possible interpretations are: "each physical unit is connected to a distinguished control unit" and "all the physical units are connected to a common control unit", formalized, respectively, by:

$$\forall x.\exists y.(connected(x, y) \wedge \forall r, s.((connected(r, s) \wedge x \neq r) \rightarrow y \neq s))$$
$$\exists y.\forall x.connected(x, y)$$

4.2 Formalising disjuncion, weakness and temporal ambiguities: CTL

We consider a *branching-time* temporal logic that is a subset of CTL - Computation Tree Logic [8] [4]:

$$\phi ::= \ | \, true \, | \, p \, | \, \neg\phi \, | \, \phi \vee \phi \, | \, AX\phi \, | \, EX\phi \, | \, AF\phi \, | \, EF\phi \, | \, AG\phi \, | \, EG\phi$$

Each CTL operator is a pair of symbols. The first one is either A ("for All paths"), or E ("there Exists a path"). The second is one of X ("neXt state"), F ("in a Future state", i.e. in a state of the path), G ("Globally in the future", i.e. in all states of the path).

Req 33 "as soon as the water measuring unit is repaired, the program returns into mode degraded or into mode normal. "

This requirement is ambiguous because of the presence of *as soon as* that can be interpreted as *next step* or as *eventually* and formalised accordingly as follows:

[4] the definition we provide is not minimal, and some operators can be derived by other. We use this definition for simplicity and readability.

$AG(water\_munit\_repaired \rightarrow AX(mode\_degraded \vee mode\_normal))$
$AG(water\_munit\_repaired \rightarrow AF(mode\_degraded \vee mode\_normal))$

The formalization of the disjunction *or* makes non-determinism explicit: this step makes the analyst aware of the condition, if non-determinism is intended, then the specification itself will contain a disjunction, otherwise the requirement must be disambiguated.

Req 17 "as soon as the water level is below N1 or above N2 the level can be adjusted by the program by switching the pumps on or off. "

Besides the temporal ambiguity, that we interpret here as *next step*, there is a weak verb, namely *can*. As in the previous case, if the use of a weak verb is intentional, then the specification will take this into account, using an appropriate modality:

$AG((water\_below\_N1 \vee water\_above\_N2) \rightarrow EX switched\_pumps)$

otherwise the requirement is disambiguated, for instance, imposing to switch the pumps:
$AG((water\_below\_N1 \vee water\_above\_N2) \rightarrow AX switched\_pumps)$

Note also that, for more precision, the formula can be split to indicate the exact action taken in the two cases of low and high water, respectively:

$AG(water\_below\_N1 \rightarrow AX switched\_pumps\_on)$
$AG(water\_above\_N2 \rightarrow AX switched\_pumps\_off)$

Req 37 "This mode can also be reached after detection of an erroneous transmission between the program and the physical units. "
There are three defects in this requirement:

This, that is a false positive since it is easily disambiguated reading requirement 36: it refers to the emergency stop mode;
can as above, the weakness is left in the formalization, using path quantifier "E", or removed using modality "A".
after: it is not clear if it refers to the next state or to a future one, hence either "F" or "X" are to be used.

Metric temporal logic Also, when specifying requirements with temporal expressions, the analyst's first decision is whether to model them in terms of computational steps (LTS) or use a real-time model. For example, *as soon as* can be the next step or within a time limit expressed in seconds or milliseconds. Requirement 33 "as soon as the water measuring unit is repaired, the program returns into mode degraded or into mode normal " for instance could be interpreted as:

"within n millisecond after the water measuring unit is repaired, the program returns into mode degraded or into mode normal. "
and specified in a metric temporal logic [11].

5 Conclusions

In this paper we have briefly shown an approach to address the problem, often discussed by Egon Boerger, of grounding the software development process on a formal basis able to capture the intended behaviour by avoiding the typical ambiguity that natural language requirements inherently contain. The presented approach follows a different research path with respect to the one mastered by Egon, in which a ground model by means of ASM is developed first with the aim of solving potential ambiguities. We focus instead on the ambiguity sources in the natural language expression of requirements, exploiting a NLP tool to point out potential ambiguities, then recurring to focused logic formulae expressing the different interpretations that may solve them.

Acknowledgements

The research has been partially supported by the project "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems), MIUR PRIN 2017FTXR7S .

References

1. Jean-Raymond Abrial. Steam-boiler control specification problem. In Abrial et al. [2].
2. Jean-Raymond Abrial, Egon Börger, and Hans Langmaack, editors. *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grew out of a Dagstuhl Seminar, June 1995)*, volume 1165 of *Lecture Notes in Computer Science*. Springer, 1996.
3. Daniel Berry, Erik Kamsties, and Michael Krieger. From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0. 2003.
4. Egon Börger. Why use evolving algebras for hardware and software engineering? In Miroslav Bartosek, Jan Staudek, and Jirí Wiedermann, editors, *SOFSEM '95, 22nd Seminar on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 23 - December 1, 1995, Proceedings*, volume 1012 of *Lecture Notes in Computer Science*, pages 236–271. Springer, 1995.
5. Egon Börger. The ASM ground model method as a foundation for requirements engineering. In Nachum Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.
6. Egon Börger. The abstract state machines method for modular design and analysis of programming languages. *J. Log. Comput.*, 27(2):417–439, 2017.
7. Gobinda G. Chowdhury. Natural language processing. *Annu. Rev. Inf. Sci. Technol.*, 37(1):51–89, 2003.

8. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.
9. Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. An automatic quality evaluation for natural language requirements. In *Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 1, pages 4–5, 2001.
10. Stefania Gnesi, Giuseppe Lami, and Gianluca Trentanni. An automatic tool for the analysis of natural language requirements. *Computer Systems: Science & Engineering*, 20(1), 2005.
11. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real Time Syst.*, 2(4):255–299, 1990.
12. Hui Yang, Anne N. De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. Analysing anaphoric ambiguity in natural language requirements. *Requir. Eng.*, 16(3):163–189, 2011.

A Steam boiler requirements

1. The program communicates with the physical units through messages which are transmitted over a number of dedicated lines connecting each physical unit with the control unit. In first approximation, the time for transmission can be neglected.
2. The program follows a cycle and a priori does not terminate.
3. This cycle takes place each five seconds and consists of the following actions: Reception of messages coming from the physical units; Analysis of informations which have been received; Transmission of messages to the physical units.
4. In first approximation, all messages coming from (or going to) the physical units are supposed to be received (emitted) simultaneously by the program at each cycle.
5. The program operates in different modes, namely: initialization, normal, degraded, rescue, emergency stop.
6. The initialization mode is the mode to start with.
7. The program enters a state in which it waits for the message STEAM-BOILER-WAITING to come from the physical units.
8. As soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero.
9. If the unit for detection of the level of steam is defective—that is, when u is not equal to zero—the program enters the emergency stop mode.
10. If the quantity of water in the steam-boiler is above N2 the program activates the valve of the steam-boiler in order to empty it.
11. If the quantity of water in the steam-boiler is below W then the program activates a pump to fill the steam-boiler.
12. If the program realizes a failure of the water level detection unit it enters the emergency stop mode.
13. As soon as a level of water between N1 and N2 has been reached the program can send continuously the signal PROGRAM-READY to the physical units until it receives the signal PHYSICAL_UNITS_READY which must necessarily be emitted by the physical units.
14. As soon as this signal has been received, the program enters either the mode normal if all the physical units operate correctly or the mode degraded if any physical unit is defective.
15. A transmission failure puts the program into the mode emergency stop.
16. The normal mode is the standard operating mode in which the program tries to maintain the water level in the steam-boiler between N1 and N2 with all physical units operating correctly.
17. As soon as the water level is below N1 or above N2 the level can be adjusted by the program by switching the pumps on or off.
18. The corresponding decision is taken on the basis of the information which has been received from the physical units.
19. As soon as the program recognizes a failure of the water level measuring unit it goes into rescue mode.
20. Failure of any other physical unit puts the program into degraded mode.

21. If the water level is risking to reach one of the limit values Mi or M2 the program enters the mode emergency stop.
22. This risk is evaluated on the basis of a maximal behaviour of the physical units.
23. A transmission failure puts the program into emergency stop mode.
24. The degraded mode is the mode in which the program tries to maintain a satisfactory water level despite of the presence of failure of some physical unit. It is assumed however that the water level measuring unit in the steam-boiler is working correctly. The functionality is the same as in the preceding case.
25. Once all the units which were defective have been repaired, the program comes back to normal mode.
26. As soon as the program sees that the water level measuring unit has a failure, the program goes into mode rescue.
27. If the water level is risking to reach one of the limit values Mi or M2 the program enters the mode emergency stop.
28. A transmission failure puts the program into emergency stop mode.
29. The rescue mode is the mode in which the program tries to maintain a satisfactory water level despite of the failure of the water level measuring unit.
30. The water level is estimated by a computation which is done taking into account the maximum dynamics of the quantity of steam coming out of the steam-boiler.
31. For the sake of simplicity, this calculation can suppose that exactly n liters of water, supplied by the pumps, do account for exactly the same amount of boiler contents (no thermal expansion).
32. This calculation can however be done only if the unit which measures the quantity of steam is itself working and if one can rely upon the information which comes from the units for controlling the pumps.
33. As soon as the water measuring unit is repaired, the program returns into mode degraded or into mode normal.
34. The program goes into emergency stop mode if it realizes that one of the following cases holds: the unit which measures the outcome of steam has a failure, or the units which control the pumps have a failure, or the water level risks to reach one of the two limit values.
35. A transmission failure puts the program into emergency stop mode.
36. The emergency stop mode is the mode into which the program has to go, as we have seen already, when either the vital units have a failure or when the water level risks to reach one of its two limit values.
37. This mode can also be reached after detection of an erroneous transmission between the program and the physical units.
38. This mode can also be set directly from outside.
39. Once the program has reached the Emergency stop mode, the physical environment is then responsible to take appropriate actions, and the program stops.