 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Specification, Development and Assessment for Requirements based Engineering, V1

D607.011

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Specification, Development and Assessment for Requirements based Engineering, V1
Deliverable No.	D607.011
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-02-10
Contact	José Fuentes
Organization	The REUSE Company
Phone	+34912172596
E-Mail	jose.fuentes@reusecompany.com

AUTHORS TABLE

Name	Company	E-Mail
José Fuentes	REUSE	Jose.Fuentes@reusecompany.com
Juan Llorens	UC3M	llorens@inf.uc3m.es
Anabel Fraga	UC3M	Afraga@inf.uc3m.es
Manu Berg	OFFIS	Manuela.Berg@offis.de
Christian Ellen	OFFIS	Christian.Ellen@offis.de
Martin Böschen	OFFIS	Martin.Boeschen@offis.de

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.5	29.01.2014	Document Creation	all
0.6	03.02.2014	Changes after internal review (SAGEM)	Several sections
0.7	07.02.2014	Changes after external review (Alenia)	Several sections
0.8	10.02.2014	Changes after external review (EADS IW)	Several sections

CONTENT

Content of Figures

Content of Tables

Content of Appendix

No table of contents entries found.

1 Introduction

The objective of this deliverable is to define the specification to be developed in work package 6.7 for all bricks. This deliverable will evolve during time, producing at least three versions where the specification will be concretized and detailed. In order to give some context, this document will also

- Describe the state of the art
- Do a gap analysis
- Match with the user needs coming from other WP such as UC2.3 and UC2.4.

From this foundation, the bricks will be specified.

1.1 Relationship to other CRYSTAL Documents

This document is strongly related to the other deliverables of this work package, namely D607.021, D607.031 and D607.041. They describe the tools of The REUSE Company, which will be the essential framework for the bricks to be described in this document.

This document also must match with the user needs providing from UC2.3 and UC2.4.

1.2 Relationship to results from other projects

1.2.1 CESAR – Cost-efficient methods and processes for safety relevant embedded systems

The document is related to CESAR Deliverable D_SP2_R3.3_M3, because of the alignment between the CESAR “Requirements Engineering Methods” and the CRYSTAL “Requirements Based Engineering”. The references to the CCC (Correctness – Consistency – Completeness) included in this deliverable are taken from the CESAR deliverable [CESAR_CCC,2011].

1.2.2 VeTeSS – Verification and Testing To Support Functional Safety Standards

VeTeSS deliverable D4.2/5.1/6.1, named Information Management, presents the improvements of this project in the understanding and formal representation of safety requirements.

1.3 Structure of this document

This document is organized as follows:

- **Chapter 1**. Introduction

  Introduction provides an overview about the goals of the document and about the necessity of complete, correct and consistent requirements.
- **Chapter 2**. State of The Art

  The CESAR project was a predecessor of CRYSTAL and offers criteria for completeness, correctness and consistency as well as methods like boilerplates and contracts. The REUSE Company, with the help of Carlos III University of Madrid, developed tools (combined into RQS) for ontology-based requirements engineering. Those both approaches will be described as state of the art and as starting point for the work of this WP.

- **Chapter 3**. Gap Analysis
  In this section needs are identified from the industrial partners and a brief coverage analysis for existing technology/industrial needs is produced.
- **Chapter 4.** Plan for CRYSTAL
  After having identified the gaps of the already existing tools and methods, advancements to be gathered by the CRYSTAL project are constituted.
- **Chapter 5**. Terms, Abbreviations and Definitions
  Abbreviations and definitions.
- **Chapter 6**. References
  References.

2 State of the art

The Requirement based Engineering Work-package in CRYSTAL considers the outcomes of the CESAR[1] project as the main state of the art background (to be precise, the definition of the CCC). In this section, we will introduce the main relevant requirement engineering concepts as well as tools implementing these concepts. In addition, the CESAR CCC criteria are introduced as a basic criteria catalogue for the analysis techniques and metrics which will be defined and implemented during the CRYSTAL project.

2.1 Requirements Engineering Concepts

2.1.1 Introduction

During the previous years, the CESAR project developed a very advanced approach (and also technology to some extend) that could be used to measure the quality of requirements specifications. At the same time The REUSE Company (REUSE) developed its own advanced solution for the same problem, which was commercialized in the market (RQS).

The CRYSTAL project has the goal, among others, to extend the existing technology regarding requirements specification towards a deeper CCC approach (Correctness Consistency and Completeness). All of that on top of The REUSE Company's Requirements Quality Suite (RQS) and connected to the CRYSTAL IOS (interoperability Services). The RQS would provide a set of advanced quality and retrieval services, but also would be able to get requirements (or other related assets) through the IOS as well.

Therefore, one of the first activities to be solved must be the clarification of visions and approaches, between the CESAR vision and REUSE's vision what will, in turn, build the ground of the RBE's approach in CRYSTAL. This approach, is not just focused on improving the quality of a requirements specification, CRYSTAL's RBE is also aimed to provide some sort of semantic retrieval and reuse capabilities for requirements.

Three main approaches have been envisaged to leverage the Requirements based Engineering in CRYSTAL. These approaches are:

- Ontology based approach
- Boilerplate approach
- Pattern approach

This section describes all those approaches and how to combine them together to reach the main goals of the Work-package.

The figure below shows how tools (bricks in CRYSTAL) and approaches combine together to get those goals.

[1] http://www.cesarproject.eu/

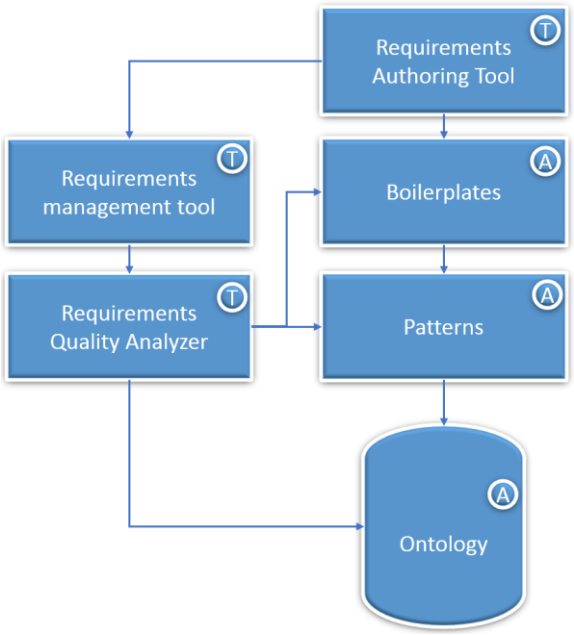


Figure 2-1. RBE Tools and approaches

Tools are shown with a (T) icon, while approaches are shown with an (A) icon.

2.1.2 Ontology

Ontologies and its applications to artificial intelligence and computer science, come from long ago. In that period, different definitions and languages have being released. There are several differences between the classical definitions of ontologies by Gruber (Gruber, 93) or Guarino (Guarino, 95) and the ontology approach envisaged in CRYSTAL RBE. Classical definitions were mainly introduced for general purposes, and represented ontologies as *a specification of a conceptualization*, that is, a set of classes, attributes, instances and links that represented a specific domain, together with some axioms and rules describing the behaviour of the represented entities.

Nevertheless, the RBE approach is pretty much focused on Requirements Engineering, Requirements Authoring and Requirements Quality what, as introduced along the document, implies several differences regarding ontological modelling.

The specialization of the ontology approach to the requirements engineering process doesn't mean that such an ontology could not be used out of the requirements domain. In fact, some of the goals of this WP have to do with enhancing the quality of SysML and UML models through a completeness and consistency analysis among requirements and models. But, ontologies can (and must) be the central point for a consistent naming all along the project.

In the RBE's approach, an ontology is made up of different layers, each using the information from inner layers.

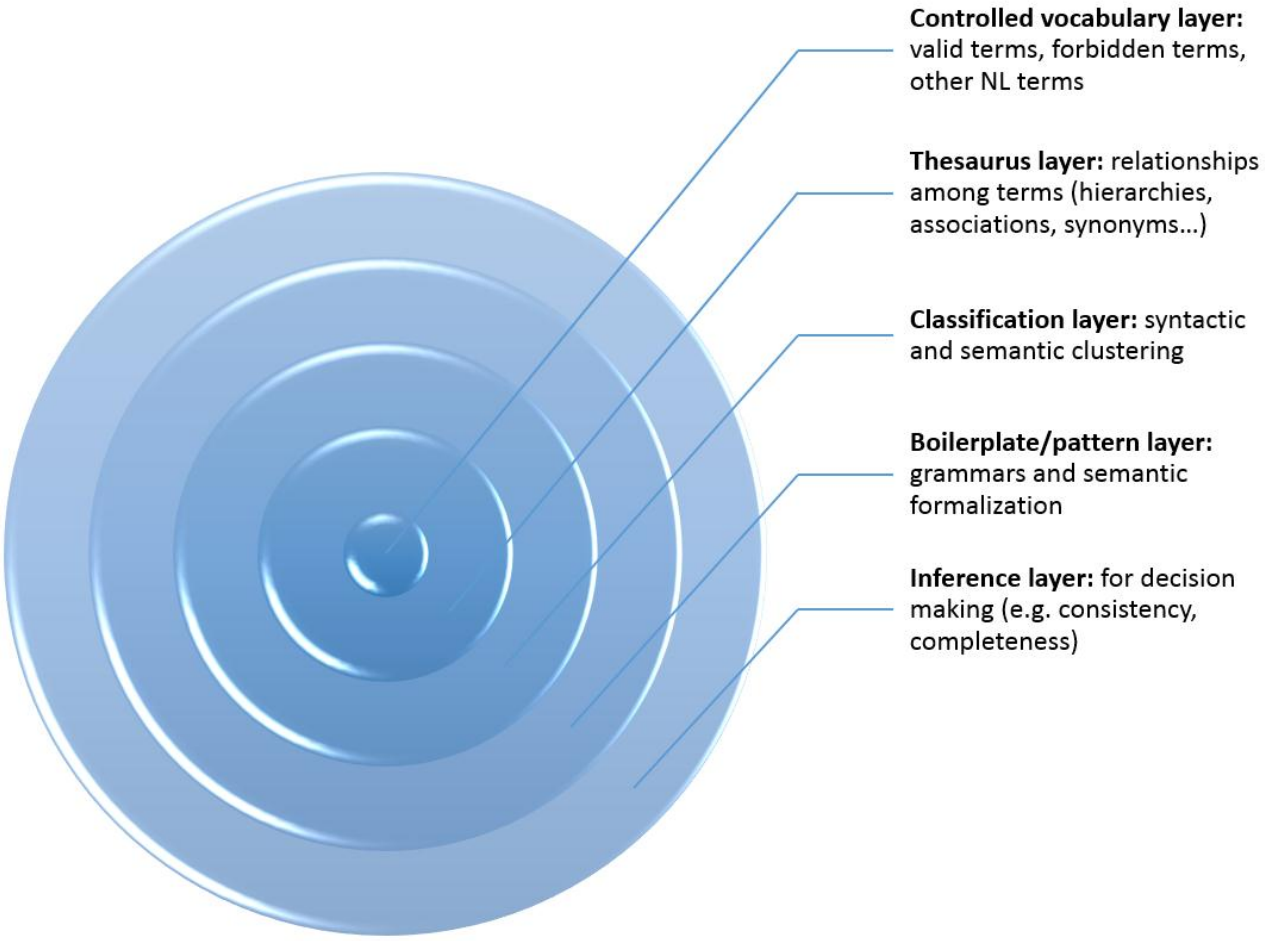


Figure 2-2. Ontology layers

- **Controlled vocabulary layer.**
  The controlled vocabulary contains a concrete and clear representation of all those concepts (represented as terms) with a specific meaning in a human language for a particular domain of application. Several types of concepts are considered.

  - **Domain concepts:**
    those concepts and names representing the domain entities
  - **Forbidden terms:**
    those terms identified as forbidden in the correctness analysis of the requirements (e.g. ambiguous words…)
  - **Other Natural Language terms:**
    unlike other classical ontology approaches, terms identified as *stop words* (e.g. prepositions, articles…) are important in this approach since they are used as part of the boilerplates/patterns and could even change the meaning of a requirement

- **Thesaurus layer.**
  For those concepts deemed as part of the business domain, some semantic links can be created. From the classical representation of thesaurus (ISO2788:1986, ISO25964-1:2011 and Z39.19:2005) come some relationships such as synonym (a very special relationship optionally including many

sub-types), hierarchical (also with many sub-types, some of them such as instance of whole-part deemed as a different type of relationship in this WP) and associative relationship. Nevertheless, the RBE's ontology approach also allows room for other types of relationships pretty much connected to the business domain (e.g. *drives, flies, manages*…)

- **Classification layer.**
  This layer allows the semantic grouping of terms assuming a "IS-A" type of relationship. It offers a multi-purpose and multi-dimensional group of clusters such as:
  - **Syntactic clustering:**
    Identifying and grouping syntactically elements such as verbs, nouns, articles…
  - **Semantic clustering:**
    Grouping together those nouns or verbs with a similar meaning. For example, as weapons we could group the terms gun, missile, mine, etc., or in the case of verbs, to drive, to pilot, to manage… all of them could share a 'similar' meaning and thus, could be grouped into the same semantic cluster. This grouping is more or less the classification fundamental of the ontologies (class). As the terms share the same nature, therefore they can be used in the requirements patterns in a similar way
  - **Relationship types:**
    The typology of the relationship is defined in this classification layer.

- **Boilerplate/Pattern layer.**
  By means of a series of slots, the agreed upon grammars for requirements (or other natural language assets along the PLM – Product Lifecycle Management) is represented together with its semantic formalization

- **Inference layer.**
  This layer intends to represent the proper way for consistency and completeness checking among requirements or even among other assets such as SysML/UML models. The definition and development of this layer will be an objective of CRYSTAL.

2.1.3 Boilerplates/Patterns

2.1.3.1 Introduction

Natural Language is still the mechanism used in most specifications to write requirements, mainly due to its unbeatable communication property: natural language is by far the best human communication language and requirements have a clear communication role. The author states his/her needs in free text (free Natural Language) because it provides an almost unlimited set of resources. However, it also allows room for ambiguity and many other well-known bad habits (collected in the CESAR's CCC approach, but also in other resources such as the INCOSE Guide for Writing Requirements [INCOSE]).

Boilerplates or pattern-based authoring tools offer authors a set of agreed-upon set of grammars for concise requirements, thus directly avoiding some of the most common mistakes related with natural language requirements, and easing the analysis for some other mistakes[2].

[2] The full set of CCC criteria covered in CESAR and CRYSTAL is depicted in further sections of the deliverable.

2.1.3.2 CESAR's view

One of the most relevant contributions of the CESAR project to the Requirements Engineering community has been the approach produced to work with requirements quality management. In this approach, the requirements are transformed from "not controlled" natural language to "instantiated and completely controlled" structured text. The transformation process includes an intermediate step, where the structure of the requirement is defined. In the next picture, Dr Bogusch (from AIRBUS) presented the CESAR vision.

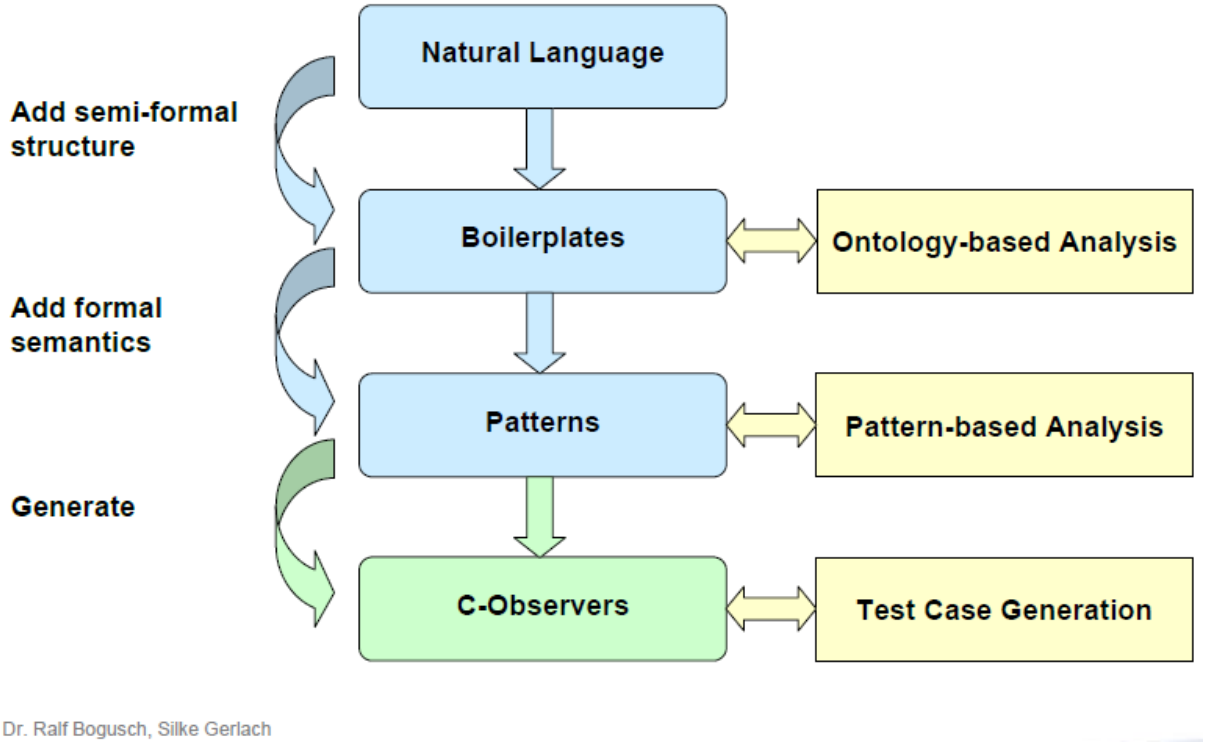


Figure 2-3: Transformation of Requirements in CESAR

The CESAR approach aims to define a way to formalize requirements by refining the operation with raw text requirements up to a "formal" requirement representation, named pattern. Indeed, its further refinement could even produce test cases generation, named C-Observers.

In order to produce the different refinements, a Domain Ontology is created and used (see Figure 2-4).

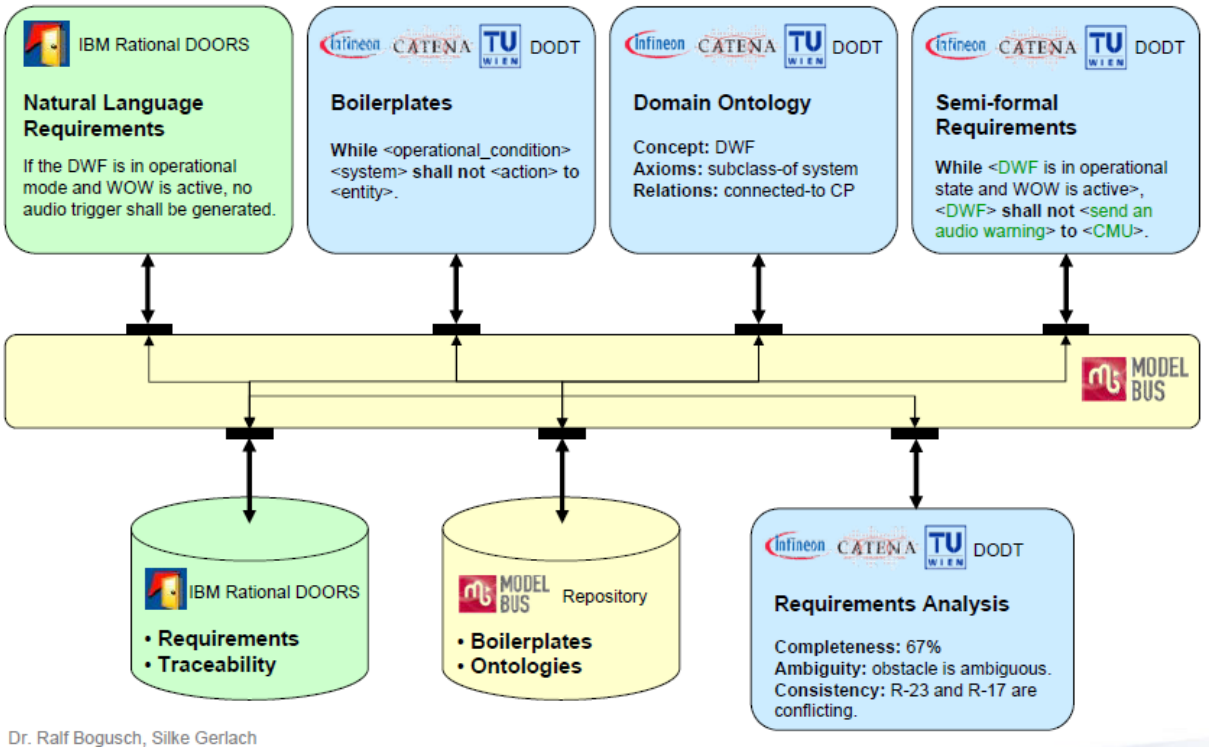


Figure 2-4: Domain Ontology in Cesar

The traversal from text requirements to formal requirements is made by attempting to produce a transformation on the requirement text itself. The final version of the requirement, considered “formal” in CESAR, is somehow instantiated from a clearly defined syntactic + Semantic structure called Pattern. The requirement is, therefore, “instantiated” from the pattern.

This slide is collected from Ralf Bogusch presentation of CESAR. Requirements formalization using boilerplates and domain ontologies (July 2011)

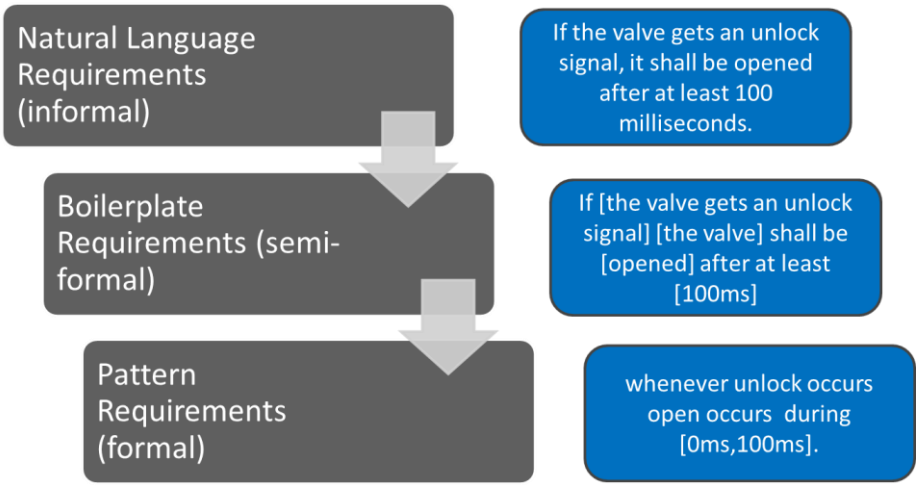


Figure 2-5: Transformations in Cesar

In this approach, a Boilerplate is simply a textual template for a requirement, as Hull, Jackson and Dick describe in their book [Hull et al 2002].

According to the RSL CESAR report [CESAR_RSL,2011], knowledge is represented in an ontology from which a variety of statement-level templates could be generated by traversing parts of the structure as appropriate. The nodes are typed entities that form the placeholders, and the arcs describe the relationships between those entities.

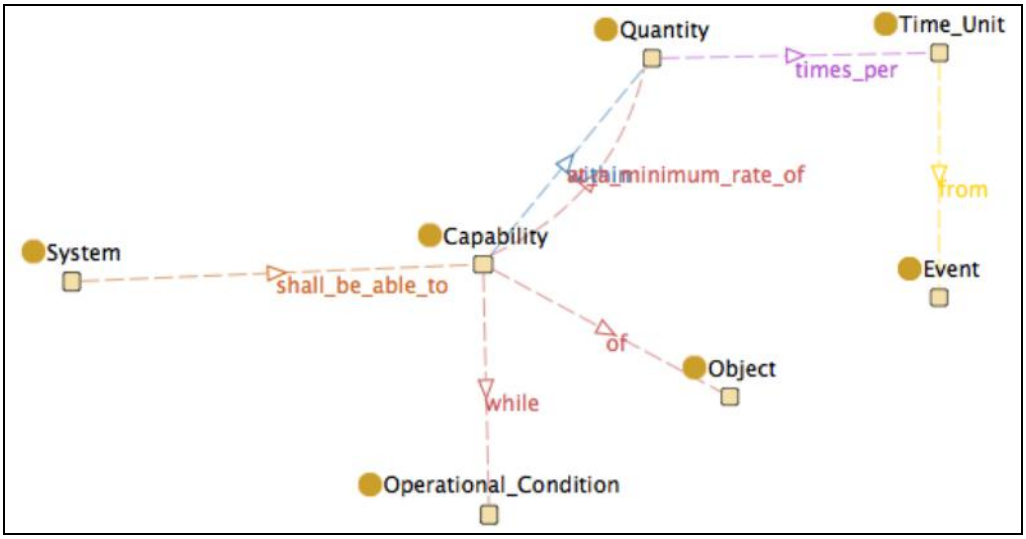


Figure 2-6: Example ontology underpinning requirement statements

OFFIS, as a CESAR partner, implemented the CESAR requirement process by using the requirement specification language (RSL) for capturing formal requirements [CESAR_RSL,2011]. The RSL defines a variety of patterns. Patterns consist of static text elements and attributes being filled in by the requirements engineer. Each pattern has a well-defined semantic in order to ensure a consistent interpretation of the written system specification across all project participants. Even though on one hand this limits the possibilities of writing a requirement, but on the other hand it prevents misunderstandings regarding the interpretation of the sentences. In CESAR's view, to gain a set of unambiguous requirements this limitation is necessary. However writing requirements shall still be possible. Patterns allow the writing of natural sounding requirements with defined semantics while being expressive enough to formalize complex requirements.

To gain a high degree of intuitivism the language consists of only a few constructs that can be easily remembered to give the requirement engineer the possibility to fully understand the RSL and choose the right pattern that fits the properties he wants to demand on the system.

There are existing patterns for each of the following categories:

- **Functional Patterns**
  These Patterns express functional requirements of the system. This includes the relationship between events, handling of conditions and invariants and the possibility to define intervals in which the requirements or parts of them are valid.
- **Probability Patterns**
  In nearly all safety critical systems, it is necessary to express failure or hazard probabilities. Since there are various partly redundant forms of expressing probabilities that were used quite ambiguous in common speech, these patterns guide the requirements engineer to express his/her needs while reducing ambiguity.

- **Safety related Patterns**
  Only a small part of safety related requirements can be covered with probability patterns. The major part of the requirements outlines relationships between system components. These patterns enable the specification of single point of failures or other failure and hazard dependencies between different components.
- **Timing Patterns**
  These patterns can be used to describe real-time behavior of systems. This includes periodic and aperiodic activations, jitter or delay.
- **Architecture Patterns**
  These patterns specify the existence of architecture elements like components, events or connections between components.
- **Mapping Patterns**
  These patterns can be used to express requirements on the mapping from the functional design perspective to the physical design perspective.

Patterns are noted in a form with optional parts that are not necessarily instantiated. A functional pattern describing the causality between two events does look like this:

Whenever *request* ***occurs*** *response* ***occurs during*** *[0ms,10ms].*

Phrases in square brackets are considered optional, bold elements are static keywords of the pattern and italic printed elements represent attributes that have to be filled out by the requirements engineer. When filling the attributes there are no general restrictions on the names but they should be descriptive. Concrete implementations of specification and analysis tools may introduce additional restrictions. The names of the attributes can be used later to be mapped to a given architecture or they can be used to generate the architecture. When specifying requirements, there is no strict typing like *int* or *real* but there are categories that specify what is described by the different attributes.

In order not to complicate the language too much, but also to be expressive enough, the set of supported attributes has to be carefully chosen:

- **Event**
  Events represent the activity in the system. An event can be for example a received signal, a user input by a pressed button, a computational result or the change of a value. Events occur at one point of time and have no duration in time.
- **Condition**
  A condition is a logic and/or arithmetic expression based on variables and the status of events. Conditions can be used in two different ways in patterns. On the one hand they can be used as pseudo-events to trigger an action if the condition becomes true. On the other hand the condition can be the system state that has to hold for a specified time. The values the condition is based on may change with time only.
- **Interval**
  Intervals describe a continuous fragment of time which can have relative and quantitative time measures as delimiters or events. Intervals can be open “`]x,y[`“ or closed “`[x,y]`” or a combination of them.

- **Jitter**

  Jitter of an event is the variation in the point of time of occurrence in a real time context. In this document jitter is considered as always positive, i.e. the event is delayed.

- **Component**

  Components refer to entities that are able to handle events or condition variables. Typically these components refer to an actual architecture, but in early design stages, where the system has not been designed completely there are also “possible” component names allowed. These identifiers can either be used to generate the missing architecture or be mapped to the design in later development stages.

Based on the formal semantics of the RSL language, requirement based analysis procedures were implemented within multiple projects. Two examples are the consistency and the entailment analysis.

The consistency analysis can be used to verify that a given set of requirements is not inconsistent, which means that there is at least one execution trace out of all possible implementations of the requirements which fulfils all requirements at the same time.

The entailment analysis supports the engineer during the decomposition of a system into multiple subsystems or after a (sub-)system requirement has been changed. It performs a virtual integration test (VIT) which is only based on the contracts. The main idea is that the combined contracts of the sub-components must fulfil (imply) the contracts on the parent component.

2.1.3.3 The REUSE Company’s view: The Requirements Quality Suite (RQS)

At the same time the CESAR project advanced, The REUSE Company (REUSE) developed its own solution to manage the quality of text based requirements: the Requirements Quality Suite (RQS) was produced. The main intention of RQS was to produce a concrete measurement of the quality of a requirements specification based on metrics for Correctness, Consistency and Completeness, and using requirements patterns and formal representations of requirements as graphs.

In RQS the term *boilerplate* was used as a synonym of *pattern*, so there is no distinction between both concepts. A pattern (or boilerplate), is simply a statement level structure: a sequential list of pattern restrictions (a.k.a. slots or pattern items) at the syntactic and/or semantic level that will be attempted to be matched with the input natural language text.

In the REUSE approach, a pattern encapsulates the rules for writing and validating a text statement of a particular kind, as well as the knowledge for producing a formal representation of the requirement. In order to match a text with a pattern, a Pattern Matching process is performed. This approach, as a difference to the CESAR’s, allows the requirements authors to write the requirements in natural language with no restrictions and the tools systems are in charge to automatically identify the patterns that match the input, as well as produce a formal representation of the requirement.

Therefore, the analyst does not need to have knowledge about the patterns or the way s/he must write the requirements

The pattern and its matching process is presented in the following picture:

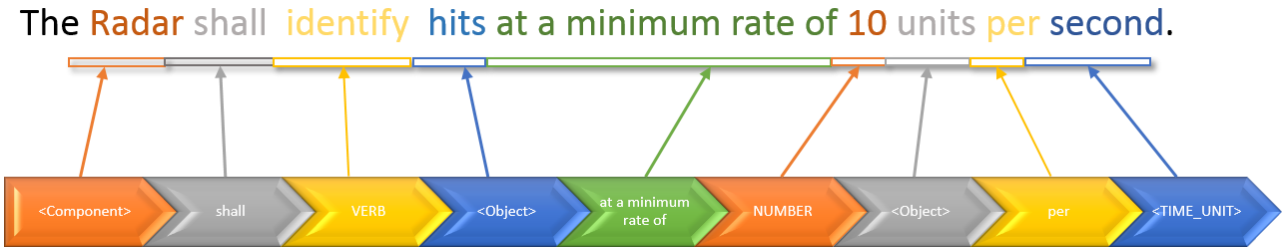


Figure 2-7: Pattern matching of terms in Natural Language text

The restrictions of the pattern can be of several types, and have different properties (optional, compulsory, OR restrictions, etc.).

The restriction types for the slots can be:

- **Term restriction:**
  A term must be found at a particular position of the requirement. Those terms are coming from the controlled vocabulary, either compound words or simple words. Represented in lowercase in the image above (e.g. *shall* or *at a minimum rate of*).
- **Syntax (the Tag) restriction:**
  A term of a determined syntax must be found at a particular position of the requirement. The tags come from the classification layer of the ontology, namely VERBS, NOUNS,… represented in uppercase in the image above (e.g. VERB, NUMBER).
- **Semantic (the Class) restriction:**
  A term of a determined Semantic must be found at a particular position of the requirement. The semantic type also come from the classification layer of the ontology. In the image above they are represented into angle brackets (e.g. <Component>, <Object> and <Time_Unit>.
- **Syntax + Semantic restriction:**
  The combination of both classification dimensions.
- **Sub-pattern restriction:**
  A determined combination of terms, matching a sub-pattern must be found at a particular position of the requirement. Most of the times, a pattern is made up of a combination of different sub-patterns. It allows the representation of whatever sub-structure, like the *Prefix* + *Main* + *Postfix*, but it's even more flexible allowing any combination of sub-patterns at any level, what means that a sub-pattern may also include sub-sub-pattern slots. This way of organizing structure allows to define requirements patterns by high level organizations. For example, the following requirement pattern:

  Whenever *request* ***occurs*** *response* ***occurs during*** *[0ms,10ms]*

  could be represented in RQS as :

  whenever *{request-sub-pattern}* ***occurs*** *{response sub-pattern}* ***occurs during*** *{interval sub-pattern}*

where the {interval sub-pattern} could be defined as:

[NUMBER <Measurement Unit> , NUMBER <Measurement Unit>]

Having

whenever, **occurs**, and **during** are terms of the ontology, as well as the symbols "[" "," "]"

The table below represents yet another example. This time, the full pattern is made up of three different sub-patterns:

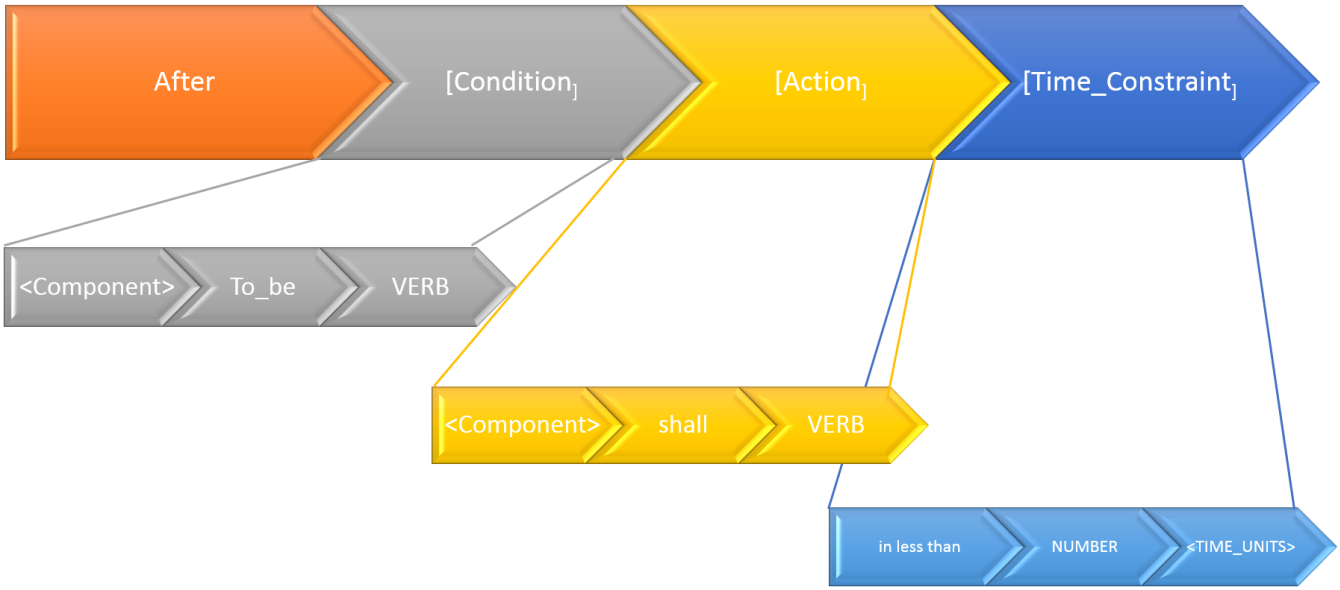


Figure 2-8. Patterns with sub-patterns

In the example above, the main pattern is made up of three different sub-patterns. The sub-pattern slots in this example are not recursively represented as sub-sub-patterns, but it could be so if needed. It's important to note, that the level of expressiveness in this approach allows, for example, that the verb in the first sub-pattern (the one as <component> + "*to_be*" + VERB) was only matched when the verb comes in a particular tense (form) in order to match the following requirement. The following example will match, since the verb *to press* is used in past tense:

After the brake pedal is pressed, the car shall decelerate in less than 1ms

While the following sentence will not match the pattern:

After the brake pedal is press, the car shall decelerate in less than 1ms

All the possible restrictions for the different slots are, as described above, elements of the ontology. Therefore, the proper ontology should be defined prior to write any requirements. The management of the patterns, as well as all the slot restrictions (terms from the controlled vocabulary, classification items…) is fully covered in the knowledgeMANAGER brick.

It's also important to note, that this is not a parameterised approach, in order words, requirements are not fully represented as instances of patterns. That means that in case the ontology managers decide to change the structure of the pattern, the requirements will remain the same.

RQS Transformation process:

In order to calculate a quality value for the requirement and the whole requirements specification, several transformations are produced to it, where the requirement is finally formalized as a graph, and operated as a graph to calculate different metrics.

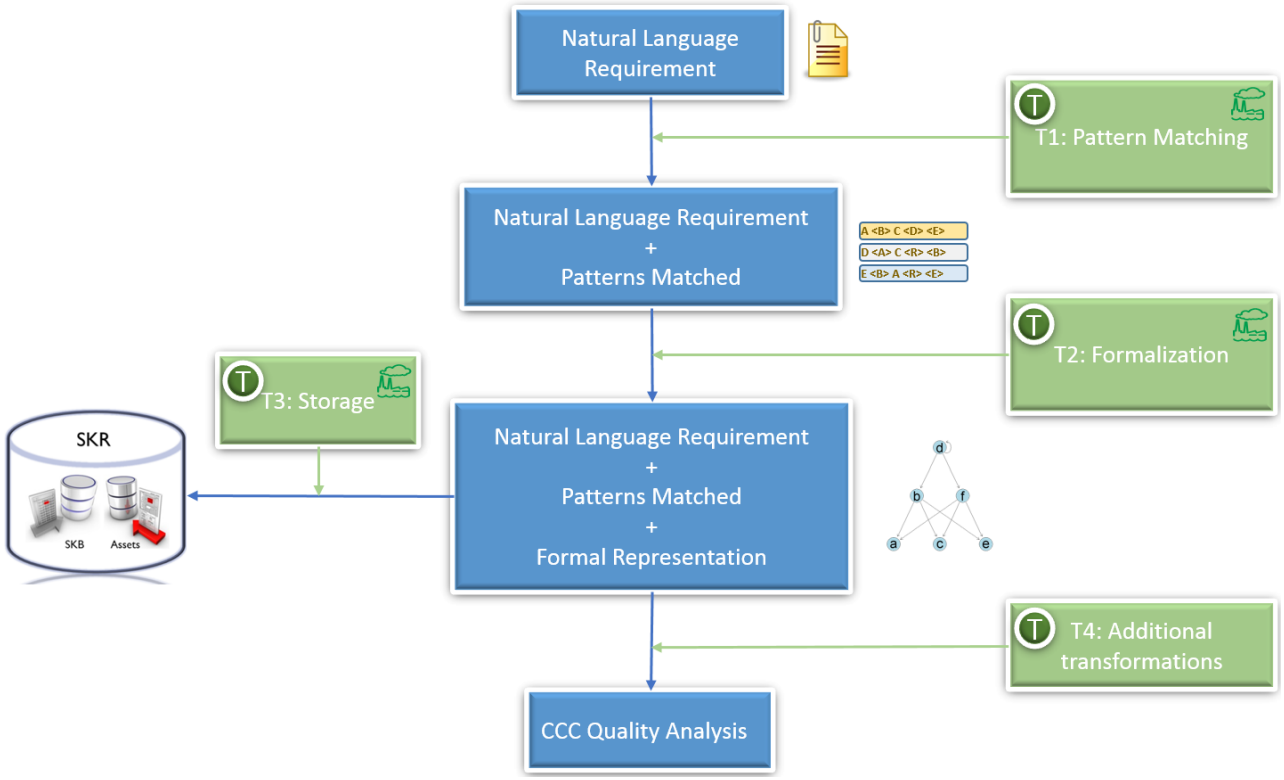


Figure 2-9. Transformations model in RQS

The different transformations are now detailed:

T1. Pattern matching:

The first operation is applied to the natural language text: Pattern matching (labelled as T1). The natural language text is analysed and pattern matching technologies are used to identify which patterns (if any) match the input text.

T2: Structural transformation

Once a set of patterns has been selected as matching the text, a new transformation operation, labelled T2 will use the information in the patterns to produce a formal representation of the natural language text using

the universal representation model RSHP [Llorens, 2004] (similar to RDF). This transformation is seen as a means to formalise a text by creating a semantic graph close to what the text states.

The Natural language text remains as the requirement body, but a graph is produced as its formal representation by defining a 3 level mapping infrastructure: Text » Restriction » Node/Relationship:

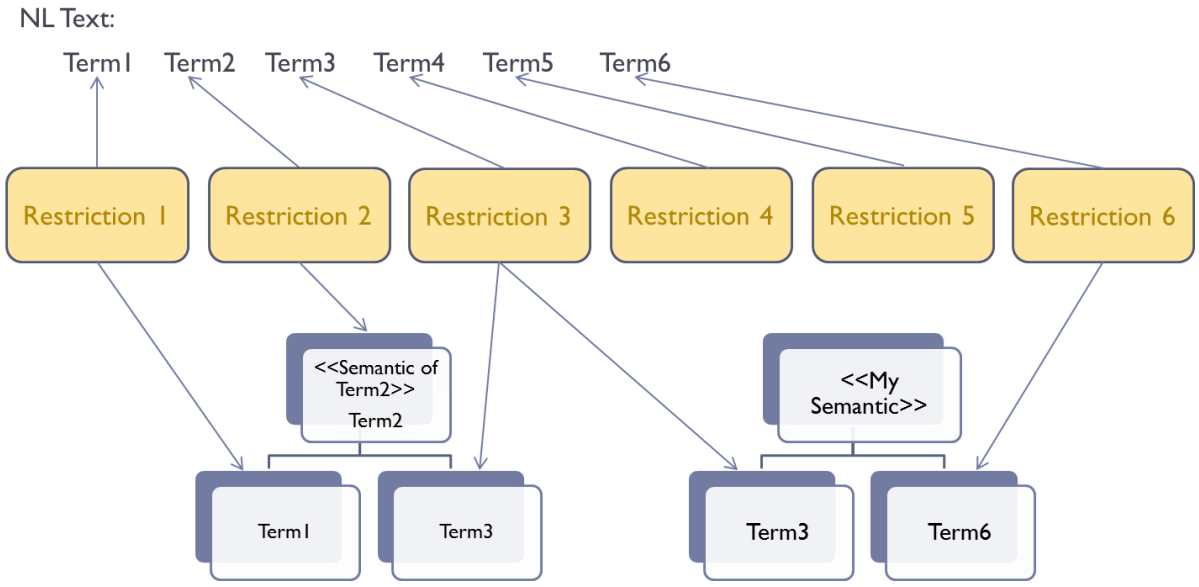


Figure 2-10. Example of formalisation

The result of T2 is a graph representation of the text, where the Nodes and arrows are clearly defined, as way as the semantics of both (Semantics of the Node = Class & semantics of the Relationship = type of Relationship)

For example: in the requirement:

> *"Whenever the pedal of the brake is pressed, the car shall decelerate immediately"*

Several possible straightforward T2 transformations of the semantic meaning of the input text could be (we show 2 different transformations):

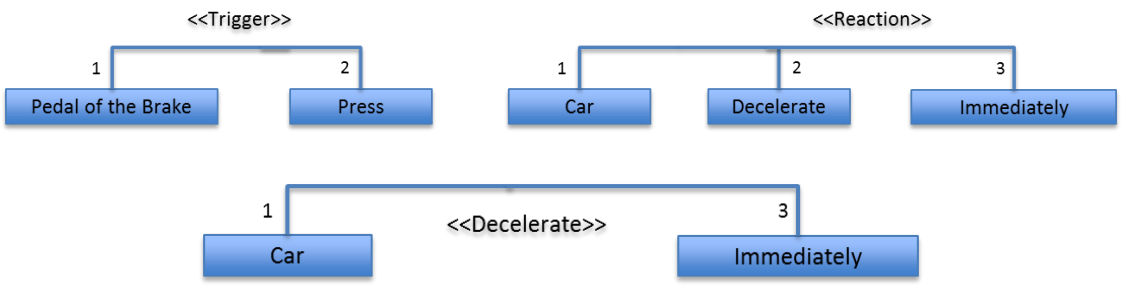


Figure 2-11. Example of formalisation

The formal representation to be produced is defined by the knowledge management department in charge of managing the ontology and the patterns.

Nevertheless, the RSHP model (and hence RQS) allows also a transformation which is not only based on semantic graphs, but also on meta-properties in form of pairs *attribute-value*. This kind of meta-properties formalization will, in most of the cases, be combined with the graph formalization approach in order to gather all the information stated in the requirement.

In the example above, aside of the arrows linking the concept of *pressing* with the component *pedal of the brake*, and the concept of decelerating with the system *car*, two more meta-properties can also be stored:

Mataproperty name	Value	Other options
When	Whenever	After, Before…
Min_Time_Constraint	0	Any number
Max_Time_Constraint	0	Any number <= Min_Time_Constraint

T2-OP: Specific transformation (OPTIONAL)

In many cases, the resulting graph produced in T2 is not enough to describe how the requirement must be represented in a formal way, as the graph produced in this transformation is describing, most of the times, the content of the requirement.

In this perception, RQS allows to perform a post-process of the formalization for customization purposes. The rules for producing a transformed graph are not coming for the matched pattern but from the ontology itself (a kind of specific knowledge about the transformation that are independent of the matched pattern) or are simple “user defined”.

Following the previous example, it could be necessary that a more formal representation must be produced for the reaction graph, where the previous graph should be transformed into:

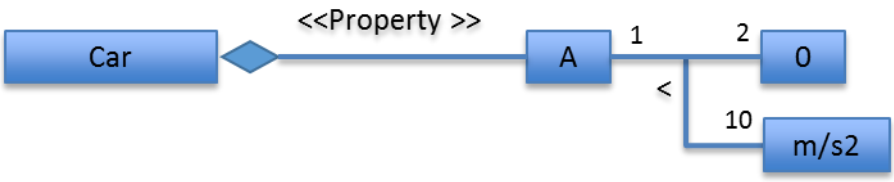


Figure 2-12. Optional T2 Transformation

where A should stand for acceleration.

This kind of transformations can be produced using a post process by applying transformation knowledge that must be coded in programming languages.

T3: Storage/Persistence

The formalization for the requirement must be stored either for further references, or for other transformations or even other analysis.

The representation schema currently followed by RQS is a relational representation of the RSHP meta-model [Llorens et al, 2004].

T4: Further additional transformations and analysis

Once the requirement is represented in a predefined way, using graphs and/or meta-properties, whatever additional transformation or deeper analysis could be possible.

Examples of such analysis can be:

- Further Correctness analysis
- Completeness analysis
- Consistency analysis

See other sections of this document to have more information about those analyses.

In addition to that, further transformations can also be envisaged:

- SysML generation: e.g. state charts
- Test case generation

In both cases, the formalized representation of the requirements should be enough to fully generate/infer SysML or test cases.

2.1.4 Contracts

The contract-based requirement specification approach extends the normal requirement specification by explicitly defining the assumptions under which the requirement has to hold. Usually, these contracts are linked to components and therefore limit the environments in which a component can be embedded.

Such a contract-based specification therefore distinguishes between assumptions on the usage context of a component and promised characteristics for the specified usage context. This is the basic principle of contract-based design approach in the SPEEDS [SPEEDS, 2008] project and of the HRC meta-model specification [HRC, 2009]. The approach allows the definition of specifications in negotiations between OEMs and component suppliers. Typically in the beginning of a development process, informal requirements are defined. Assumptions are not necessarily formulated because the system context is not yet explicitly known. When deriving requirements for parts of the system, assumptions are typically more clearly elaborated. Contract-based component specifications can be analyzed in a virtual integration test (VIT) or entailment analysis. Figure 2-13 illustrates the usage of contracts linked to components and sub-components.

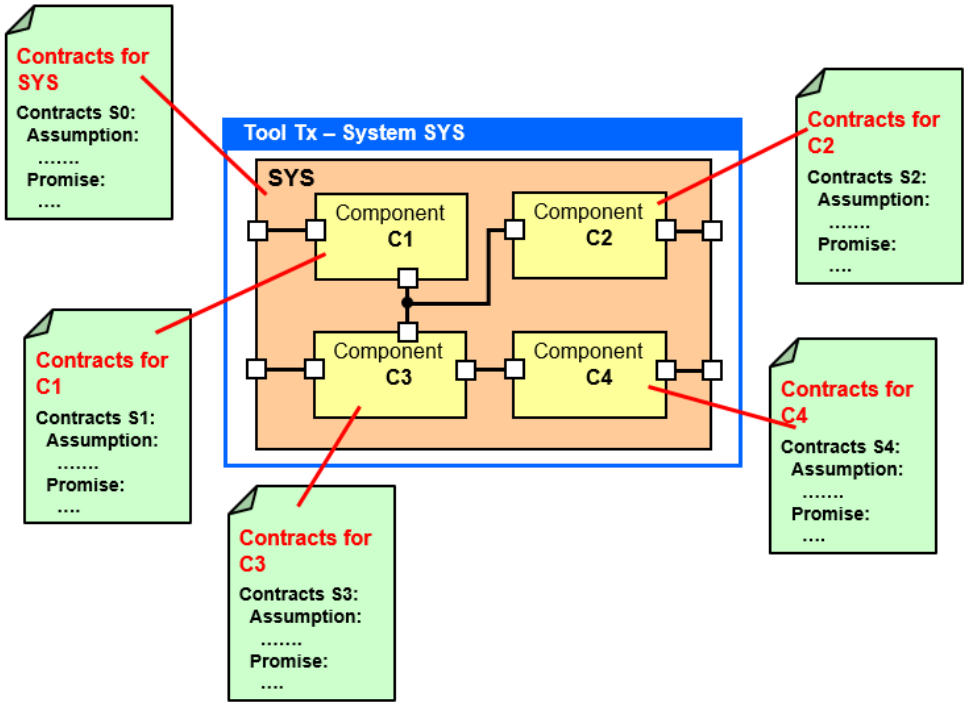


Figure 2-13: Contracts associated to components and sub-components of a system

2.2 CCC Concepts

The CESAR Project developed a catalogue of criteria. Requirements should fulfil these criteria to ensure that they are correct, complete and consistent (CCC). While Correctness criteria is assessed individually, requirement by requirement, consistency and completeness must be assessed for a set of requirements, or even for different sets of requirements or among requirements and models.

The catalogues addressed the following:

- Completeness
  - Based on viewpoint
  - higher level-requirements
  - Based on architecture viewpoint
- Correctness
  - Internal Design Constraints
  - Requirement wording
- Consistency

The main value for the CRYSTAL Project is that it provides a list, which can be checked item by item. Furthermore, specific methods addressing specific items of the list have been developed. Some of these methods can be reused in the CRYSTAL Project.

Developed methods, which address these criteria, are:

- Requirement consistency analysis
- Entailment analysis
- Interface compatibility analysis

The reader may refer to CESAR_D_SP2_R3.3_M3_Vol4_v1.000_PU.pdf [CESAR_CCC,2011]] for a more detailed survey.

As detailed later, the main differences between CESAR and CRYSTAL and, therefore, the main goal of WP607, are the following:

- CRYSTAL will address a larger list of criteria
- In CESAR, these methods were very scattered between tools, CRYSTAL will have a more integrated approach
- CRYSTAL will integrate the CCC criteria in the set of IOS services

2.3 Tools

2.3.1 DODT

The DODT tool (Domain Ontology Design Tool) as already used in the CESAR project, and allows to semi-automatically transforming NL requirements into semi-formal boilerplate requirements. The transformation builds upon a domain ontology (DO) containing knowledge of the problem domain and upon natural language processing techniques. It was developed at the TU Wien together with several industrial partners.

The tool runs several transformation steps on the requirement, which could also be seen as quality metrics. In order to produce boilerplates from the requirements, it does some transformation automatically, and other transformations are made with the help of a human requirement engineer.

Among others, the tool checks for the following things:

- if the requirement can be split into two requirements
- if there are typographic errors
- if words can be replaced by a controlled vocabulary
- if the requirement is ambiguous in the sense that several boiler plates match this requirement

Furthermore, the tool ranks different transformations by its most probable meaning, again it's done by using a domain ontology. For example, it knows that a *passenger* would be a better match for a *user* than for a *system*.

For further information, the reader is referred to [Farfeleder,2011].

2.3.2 Embedded Specifier

The BTC Embedded Specifier is a tool to formalize requirements which can later be used for automated test case generation. The tool implements a pattern-based approach and contract semantics similar to the CESAR approach. In addition, the tool also distinguished between informal, semi-formal and formal requirements.

- A semi-formal requirement is derived from a purely informal requirement by manually selecting a matching pattern from the pattern library and by identifying text blocks, so called macros, within the natural language requirement which will be used as parameters within the instantiated pattern.
- Formal requirements are refinements of semi-formal requirements which bind the macro text blocks to actual expressions on an implementation model of the component (typically Matlab SIMULINK).

The tool allows to directly generate observers for the system behaviour based on the formalized requirements.

2.3.3 Requirements Quality Suite

The Requirements Quality Suite (RQS) is a set of tools aiming to define, manage, improve, track and report the quality of the requirements specifications created for a specific project by means of the application of ontologies, patterns, natural language processing and pattern matching, as specified in the previous sections.

Those tools already address some of the CCC criteria described before (see section 2.2), mainly correctness for individual requirements. Furthermore, the suite is now partially based on the boilerplate/pattern approach defined above.

The following picture describes the three tools included in the suite, as well as a brief list of capabilities:

- **RQA – Requirements Quality Analyzer:**
  - Configure RQS with the quality policies and checklist of your organization
  - Check the correctness of your requirements specifications
  - Graphical and textual quality reporting
  - Consistency and Completeness analysis for a module or project

- **RAT – Requirements Authoring Tool:**
  - Write your requirements easily by using an assistant
  - Use the right structure and right vocabulary: patterns + ontology
  - Correctness analysis on the fly
  - Consistency analysis on the fly

- **knowledgeMANAGER:**
  - Manage all the domain knowledge behind the quality analysis
  - Management of glossaries, taxonomies, thesauri and ontologies
  - Management of the boilerplates to be used by RAT

Figure 2-14. The Requirements Quality Suite

The architecture of the suite is the following:

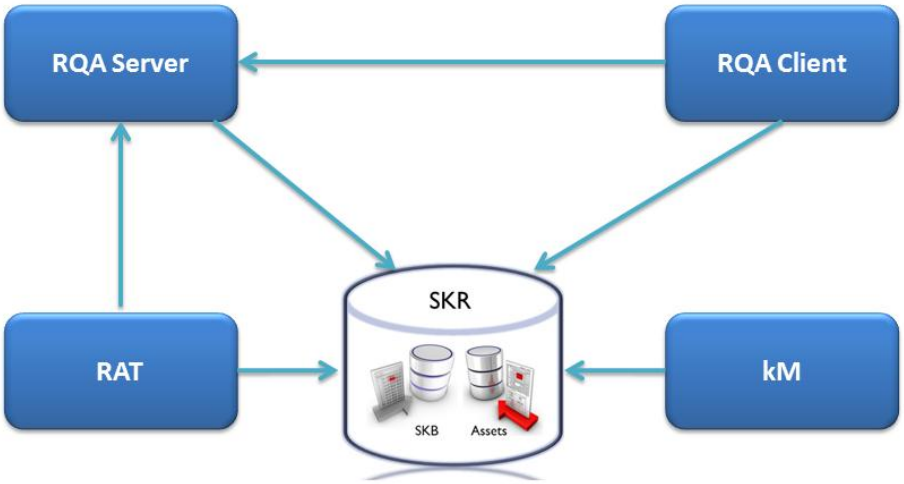


Figure 2-15. Architecture of RQS

Following this architecture:

- **RQA Server Requirements Quality Analyzer Server:**
  In charge of the main configuration of the whole suite. Database connection, licensing and low-level database management…

- **RQA Client – Requirements Quality Analyzer Client:**
  Provides the customization of the quality assessment of the suite. RQA is using the terminology provided in the various layers managed by knowledgeMANAGER, as well as the rest of the layers of the ontology managed by kM

- **RAT – Requirements Authoring Tool:**
  This module allows quality analysis on the fly, but it also includes the authoring guidance by using the information represented in the boilerplate/pattern layer

- **kM – knowledgeMANAGER:**
  Manages all the layers described in the ontology section (see section Ontology 2.1.2 above). Furthermore, it also manages all the NL tools and techniques needed for the quality analysis performed by the suite

- **SKR – System Knowledge Repository:**
  This is a relational database where two different parts can be clearly identified:
  - **SKB – System Knowledge Base:**
    Represents the main ontology behind all the quality analysis as well as all the information needed to perform a Natural Language Process to generate a semantic graph out of a textual requirement
  - **SAS – System Assets Store:**
    Includes the formal representation (as described before, in form of a semantic graph plus a set of meta-properties) generated out of every textual requirement once the requirement has been created with RAT or analysed by RQA

3 Gap Analysis

3.1 User Needs

3.1.1 Refined/Enhanced/Additional Methods for CCC

The industrial partners have asked for support to a defined set of CCC metrics:
The origins of these metrics can be found in the previous work produced by them in the CESAR project [CESAR_CCC,2011]]. These needs are very exhaustive and are presented in the following table.

Table 3-1. CCC User needs identified in CESAR

	COMPLETENESS
2.2.1.1.1	Environment viewpoint
2.2.1.1.2.1	Functional behaviour viewpoint : All prohibited behaviour characteristics are explicitly stated
2.2.1.1.2.2	Functional behaviour viewpoint :Operational scenario identified
2.2.1.1.3	Safety viewpoint
2.2.1.1.4	Process conformance viewpoint
2.2.1.1.5	Performance viewpoint
2.2.1.1.6	Production, operation, service... viewpoint
2.2.1.2.1	Requirements cover completely upper level requirements
2.2.1.2.2	All sources of requirements are identified
2.2.1.2.3	No hidden derived requirement is embedded in this requirement
2.2.1.3.1.1	Are the requirements fully allocated to architecture or components
2.2.1.3.1.2	All software requirements shall be identified
2.2.1.3.1.3	Requirements are linked to at least one lower level requirement
2.2.1.3.1.4	Interfaces issued from design have been defined
2.2.1.3.1.5	All the constraints, assumptions and derived requirements are defined, substantiated and addressed
2.2.1.3.2	Algorithms are complete and precise, including area of discontinuities and expected results are described
2.2.1.4	Based on stakeholders needs (external interfaces)
2.2.1.5	No TBDs, TBCs
	CORRECTNESS
2.2.2.1	In accordance with the upper level requirement
2.2.2.2	Realizable
2.2.2.3	Functioning modes are defined (definition of use case)
2.2.2.4	In accordance with the needs and constraints of the user
2.2.2.5.1	Derived requirements are correct and supported by analysis
2.2.2.5.2	Assumptions are identified (documented, traced…)

2.2.2.5.3	Interfaces
2.2.2.6	The requirements comply with the safety analysis
2.2.2.7	Statements leading to appropriate levels
2.2.2.8.1	Syntax
2.2.2.8.2	Atomic requirements
2.2.2.8.3	What is required - as opposed to how it should be designed, excepted for justified design constraints
2.2.2.8.4	Unambiguous
2.2.2.8.5	Verifiable
2.2.2.8.6	Maintainability of the requirement (easy updating)
2.2.2.9	Errors of fact
	CONSISTENCY
2.2.3.1	Requirement does not conflict with other and with higher requirements
2.2.3.2	Redundancy
2.2.3.3	Necessary
2.2.3.4	Combination
2.2.3.5	General

The first column presents a reference to the need in the CESAR document.

3.1.2 Tool Needs for the Requirements Quality Suite

The Requirements Quality Suite, from The REUSE Company, must provide the common platform that will solve the user needs, according also to other requirements such as the TRL and the connection to the CRYSTAL interoperable platform (IOS). Due to this requirement, the industrial partners have proposed a set of needs that must be applied to the RQS tool set; those other needs go beyond the scope of CCC needs. The main improvements to RQS are considered to be the following:

Regarding RQS Interoperability with the whole Systems Engineering eco-system

- Integration in the overall tool chain through the IOS

Regarding Ontology Management

- Import/export of ontologies
- Smart merging of ontologies
- Support to ontology evolution: not only based on the current suggestion system, but with the aid of a frequency analysis
- Ontology baseline system

Regarding Ontology Representation

- Support to PBS (Product Breakdown Structure)

Regarding Support to Supply Chain

- Provide collaborative management of ontologies along the supply chain

Regarding the whole Quality Management Process

- Pre and Post-analysis code: this represents a way for the end-users to write customized code to be executed at different particular moments while the quality analysis is performed:
  - Before the analysis: the code will be able to change any of the attributes of the requirement
  - After the analysis: RQA will provide information related to the result of the analysis so that the proper actions could be taken

Regarding Completeness

- Support to a set of new techniques for completeness checking, mainly those described in the CESAR CCC approach. Section 4 describes a set of methods identified to address this need.

Regarding Consistency

- Support to a set of new techniques for consistency checking, mainly those described in the CESAR CCC approach. Section 4 describes a set of methods identified to address this need.

Regarding Correctness

- Support to new in-built correctness metrics:
  - Deprecated concepts
  - Use of not preferred concepts (synonyms)
  - Use of concepts identified as ambiguous because of their list of more specific concepts in the ontology, in other words, in those cases where a concept has been decomposed in the ontology (either by using the hierarchical parent-child relationship or even the whole-part relationship), this concept could be deemed as ambiguous since a more specific concept should be used instead
- Customized metrics: allowing the end-user (or any other third-party) to write the code for her/his own metrics
- Enhance the current in and out-links metric with nominal links where the user could identify the name and direction of the link to quantify

Regarding RQS Configuration

- Enhance collaborative work with RAT, defining configuration options for RAT execution

Regarding reuse capabilities in RAT

- Identification of similar requirements (through their semantic graphs) in previous projects
- Identification of similar requirements within the same project or within the same requirements document

3.2 Motivation for additional concepts

There are some ideas and concepts which could help to improve the requirements' quality and, more exactly, follow the CCC criteria already described. The following section includes the ones that will be considered into the RBE Work package.

3.2.1 Improve the CESAR Requirements Engineering Approach

As described in this document, the requirement formalization process defined by CESAR consists of two independent formalization steps which refine a requirement written in natural language to a so called semi-formal boilerplate and then to a formal requirement.

Table 3-2. Example of a CESAR requirement formalization

Natural Language Requirement	
Example	If the pedal of the car is pressed, the brake force shall be applied after at most 10 milliseconds.
Structure	<English text>
CESAR Boilerplate	
Example	If [the pedal of the car is pressed] [the brake] shall [apply brake force] after at most [10ms]
Structure	If <condition> <system> shall <action> after at most <time>
Formal Requirement Pattern	
Example	Whenever (pedal_pos<100) occurs (brake_torque > 0) occurs within [0ms, 10ms].
Structure	Whenever <event> occurs <event> occurs within <time-interval>.

Several deficits of this approach have been identified and are planned to be addressed by the CRYSTAL requirement engineering approach.

First of all, the process needs a requirement engineer to be an expert in all three languages. Writing a requirement in (more or less) natural English may be imprecise and ambiguous but can be performed by most engineers by following some given guidelines. The boilerplate refinement adds the benefits from using elements of a pre-defined domain ontology, but requires the user to pick the correct pattern manually and identify the domain elements. The same argument holds for the second formalization step.

In addition, the refined versions of the requirement are harder to understand for a non-expert because the defined structures not always allow writing correct English sentences.

Having three different versions of the same requirement also increases the complexity of traceability between these requirements.

Within the CRYSTAL requirement based engineering approach we want to address these issues without loosing the benefits. To be more specific, we want to use the RQS boilerplates/patterns with their more advanced ontology features to directly infer the formal requirement based on the boilerplates. In addition, the metrics system of RQS will be used to support/guide the user writing natural language requirements, s.t. a boilerplate can be matched by the system.

This effectivly reduces the number of refinement steps necessary to reach a requirement which has a clearly defined formal semantic.

3.2.2 Patterns Instantiation

As discussed in the previous section, one core idea is to reduce the number of formalization steps needed and still end up with a requirement which has a clearly defined semantic and can therefore be used in different kinds of formal CCC methods.

The plan is to implement an automatic instantiation of formal RSL patterns based on the RQS boilerplates and additional concepts defined in the ontologies managed by the tool knowledgeMANAGER of The Reuse Company.

Depending on the kind of analysis used, the RSL patterns offer different ways to express their semantic. Typically, representations using timed automata observers or LTL (linear temporal logic) formulas are used. If a boilerplate in RQS can be matched, the idea is to instantiate a corresponding RSL pattern. Two key problems have to be addressed to enable such an automatic instantiation.

- First, the correct pattern has to be identified based on the entered natural language requirement. This can be addressed by defining good boilerplate rules in RQS which allow a high degree of freedom in the phrasing of the requirement but also guide the user by different metrics to formulate the requirement precisely.
- Second, the parameters have to be extracted correctly from the requirement. Once a parameter has been matched to a boilerplate rule, the underlying semantic model has to be extracted. Our idea is to generate an RSHP instance [Llorens et al, 2004] representing the requirement which uses knowledge about the structural information of the system defined in the domain ontology.

3.2.3 Product Breakdown Structure (PBS)

There is often a hierarchical model of the system under development. This model contains a lot of information, which can be used to improve the requirements' quality. For example, if a requirement implicitly assumes a whole-part relationship between two artefacts, it could be checked if this assumption is consistent with the PBS. Furthermore, the PBS is essential to enable an automatic pattern instantiation.

Therefore, aside of the current whole-part relationship that, among others, can be managed using knowledgeMANAGER, the whole RQS suite will provide support to PBS, not only representing the structural information of the system, but also allowing room for other kind of data that could be used during the consistency or completeness analysis.

3.2.4 Contracts as additional potential for CCC analysis

Contracts, as explained in 2.1.4, address in some points a very similar problem as a CCC analysis. For example, they are defined for a specific environment, which are checked by the completeness analysis. Using the promises of the contracts, there are potentials for checking correctness. A lot of different correctness metrics boils down just to checking the assumption of the contract. As they make assumptions and promises explicit, they allow formal reasoning about requirements. As contracts are linked to the architecture, one can check if the requirements are completely allocated to the architecture. Furthermore, contracts allow performing an entailment analysis.

3.2.5 Probabilistic requirements

By including probabilism in requirements, the expressiveness can be increased to capture events or actions which only occur with a certain probability or probabilistic rate over time. It allows modelling uncertainties within a system, e.g. a sensor will measure the correct value only with a given probability and may deliver a wrong value in the other cases.

In general, probabilistic influences within requirements can be used in different ways:

- **Assumptions on the inputs/environment:**
  Using the contract-based design principle, assumptions can be used to define the environment in which a component can be embedded. This can refer to the occurrence of events or the likelihood of external conditions. For most relevant events which are outside of the system, a probability measure can be defined to model the rate or probability over time of these events. Examples are measurement errors, system failures or weather conditions.

- **The guaranteed behaviour of a component:**
  Alternatively, a requirement can quantify its defined behaviour using probabilities or probability distributions. This can range from an overall probability that the requirement holds (when triggered) to full probabilistic distributions over the produced output values or computation times

As an example the whenever pattern from section 2.1.3.3 shall be used:

> ***Whenever*** *(pedal_pos<100)* ***occurs*** *(brake_torque > 0)* ***occurs within*** *[0ms, 10ms].*

Potential probabilistic influences to this pattern could be the occurrence of rare events like an emergency braking. One could define if the reaction times must be distributed differently in case of such a hard braking. This could be modelled by an additional assumption that an emergency brake happens with a certain rate.

In conjunction with this, the reaction time itself could be modelled not only by an interval, but with a distribution (like a normal distribution or an exponential distribution) which is potentially bounded by the interval (e.g. beta distributions).

Alternatively, the produced output values for the brake torque can also be quantified by a distribution.

The additional probabilistic information to the requirements can be used within CCC analysis techniques, e.g. to analyse if a system is correct with a certain probability given the individual probabilities within the requirements.

4 Plan for CRYSTAL

This chapter outlines the vision to be followed in order to achieve the project goals. The basic idea is the use of ontologies, boilerplates and patterns in more integrated way, but also to apply some of the concepts introduced in section 3.2.

First, the CESAR CCC criteria are refined to address the CCC needs in a structured way. In the second section we define the initial ideas of an integration of the OFFIS RSL pattern language into the kM and, therefore, into the whole RQS suite. Within the third subsection we will state a list of services that should be covered in the Interoperability IOS approach of CRYSTAL.

4.1 Reclassification of the industrial partners needs

This work-package has made some effort to reformulate the CCC-criteria of the CESAR Project, in order to have criteria which better correspond to implementable metrics. The following criteria explain the rationale behind the reclassification.

4.1.1 Correctness

Correctness of requirements refers either to the correct structure of the requirements, to the correct content of the requirements including their correct specification.

- Correctness by textual content
  - on ambiguity
  - Other metrics
- Correctness by structure of the requirement
  - on syntax
  - on type-correctness
- Correctness by formal representation
  - on terminology
  - on structure
  - on atomicity
  - by design
- Correctness by requirement management elements
  - on realizability
  - on verifiability
  - on detail level
  - on links

Wording

For natural language requirements authoring, requirement wording is a crucial quality dimension. We consider all aspects that are connected with the notion of well written requirement. The fundamentals of the supported metrics for this topic are found in the different writing style manuals and papers existing in the literature (e.g. INCOSE Requirements Guide [INCOSE]). Among them, we consider ambiguity, domain verbs, negations, etc. In natural language requirements, ambiguities caused by words have to be identified and highlighted to the engineer or don't have to be allowed at all. An example for an ambiguous wording is "the airbag is fired in 10ms". This could mean that the firing of the airbag happens within the next 10ms or after the elapsing of 10ms. See the current metrics in Requirements Quality Analyzer for a detailed list [RQA].

Structure

Concerning the structure we differ between syntax and type-correctness. Each requirement shall have a correct syntax. What is meant with correct is to be defined within the scope of the requirements engineering. In general, a restricted syntax has to be defined for all the kinds of requirements that a requirements engineer would like to use. If the engineer should use the logic LTL (linear temporal logic), then he/she would be restricted to the formal syntax of LTL. If the engineer should use natural language, there must be a restriction of the sentences he/she can build to get a defined and therefore correct sentence (boilerplates and patterns will be used to check this kind of correctness).

Type correctness addresses the question whether requirements are linked to their correct type. If, for instance, operational requirements must be described by state machines, then a state machine is expected as input for the requirement.

Content

Correctness of content refers not only to the structure of the requirements but to its inner content from a semantic point of view. Several options are studied.

Terminology correctness is based on the idea that an ontology exists, where concepts and relationships from the meta-model are addressed. The notion of terminology correctness measures if the terminology used in a requirement has been previously defined in the ontology.

Structural correctness refers to the assumption that concepts are not only characterized by their names but could also contain data structures with data types, structural correctness of concepts means keeping the structural needs for each concept. For instance, a rationale could contain a description while interfaces include function names and input as well as output variables. If the relationships can also have types, the criterion structural correctness of relationships is analogously defined.

Atomicity addresses the problem that every requirement should be stated atomic. It is a bit problematic to define this criterion precisely. Moreover it might be in conflict with the criterion of combination and redundancy.

Correctness by design refers to correct design decisions of the engineers. This is a criterion where formal methods cannot support, since decision should be checked by humans. For instance, demanding correct rationale would require checking the links to the rationale and deciding whether the rationale was chosen well. The last sub-category addresses algorithms. Even if they do not refer to the specification of requirements themselves, it seems plausible that an engineer might be interested in fixing a concrete algorithm within a certain requirement (although usually it is not desired to write in requirements how the problem should be solved). The given algorithm has to be complete and correct (in the theoretical sense), has to have an initialization and should be either deterministic, or in some special cases even probabilistic.

Environment

In the correctness by the environment of the requirement we consider all aspects associated with the connection of the requirement outside itself, with other system engineering artefacts or even requirements.

Realizability is a criterion which comprises all stated requirements and asks whether there is a system or component or configuration such that all the requirements will be met. It is most feasible to ask for realizability on the technical perspective.

Verifiability is a sub-category of the criteria. The criterion “classification possible” asks whether requirements are verifiable by stating whether they are testable, demonstrable, or analysable. The second criterion of “analysability in a formal model” asks given a formal model as a requirement or part thereof whether it can be analysed by e.g. checking reachability in a state machine.

A requirement is related to the appropriate level of detail if the concepts that are used in the requirements correspond to the level of detail to which also the requirements belongs to. For instance, if the requirement belongs to the technical perspective, then only concepts (words) belonging to the technical perspective and not e.g. to the functional perspective should be used.

Finally, in many cases it becomes necessary to measure the existence of different links between the requirement and other requirements, rationale, designs, documentations, specifications, etc.

The following table summarizes the metrics for Correctness that the brick partners would like to support at the end of the project. The purpose of the bricks is that the industrial partners will produce a ranked list of the metrics in their importance order. The bricks, depending on technical and skills, should try to fulfil the order.

Table 4-1. Metrics for Correctness

CORRECTNESS (C1)
A- Based on Requirement textual content
C1-A-3- Does the requirement avoid using ambiguous sentences?
C1-A-2- Do all the acronyms included in the requirement have an accepted common meaning?
C1-A-1- Does the requirement reach a suitable readability mark?
C1-A-4- Does the requirement have an accepted value of characters between punctuation marks?
C1-A-5- Does the requirement avoid using conditional tense?
C1-A-6- Does the requirement avoid using connectors?
C1-A-7- Does the requirement contain dependencies?
C1-A-8- Does the requirement avoid using design sentences?
C1-A-9- Does the requirement contain enough domain concepts?
C1-A-10- Does the requirement contain enough domain verbs?
C1-A-11- Does the requirement avoid using flow sentences?
C1-A-12- Does the requirement use imperative tense?
C1-A-13- Does the requirement avoid using implicit sentences?
C1-A-14- Does the requirement avoid using incomplete sentences?
C1-A-15- Does the requirement have in-links?
C1-A-16- Does the requirement avoid using negative sentences?
C1-A-17- Does the requirement avoid using too many nested levels?
C1-A-18- Does the requirement have out-links?
C1-A-19- Does the requirement avoid using too many paragraphs?
C1-A-20- Does the requirement avoid using passive voice?
C1-A-21- Does the requirement lack rationale sentences?
C1-A-22- Does the requirement reach a suitable readability mark?
C1-A-23- Does the requirement avoid using speculative sentences?
C1-A-24- Does the requirement avoid using subjectivity sentences?
C1-A-25- Does the requirement use a suitable text length in number of characters?
C1-A-26- Does the requirement avoid using unclassified concepts?
C1-A-27- Does the requirement avoid using unclassified verbs?
C1-A-28- Is the requirement not volatile?
B- Based on Requirement Structure
C1-B-1- Is the requirement standard according to the organization?
C- Based on Requirement formal representation
C1-C-1- Are interface definitions adequately defined, substantiated and addressed in the SKB?
C1-C-2- Is every interface requirement correct according to the SKB definitions?
C1-C-3- Is every requirement in accordance with the upper level requirements?
C1-C-4- Is every requirement Atomic?

D- Based on Requirement Management elements
C1-D-1- Is every requirement Realizable (based on pattern existence)?
C1-D-2- Is every requirement Verifiable (based on test case generation)?
C1-D-3- Is every requirement in accordance with the upper level requirements?
C1-D-4- Is every system functional requirement that will be implemented in SW satisfied by the software high-level requirements?
C1-D-5- Is every system performance requirement that will be implemented in SW satisfied by the software high-level requirements?
C1-D-6- Is every system safety-related requirement that will be implemented in SW satisfied by the software high-level requirements?
C1-D-7- Is every requirement Realizable (based on traceability)?
C1-D-8- Are all requirements linked to stakeholder needs?
C1 D-9 Based on the existence and value of an attribute (e.g. Rational)

4.1.2 Consistency

The second class of criteria is consistency. With consistency in general, we mean that there is no conflict between sets of requirements.

Another vision of consistency is whether requirements are redundant. In general, there is only a matter of expenses and inefficiency (at least it's less important than contradictory information among requirements) whether redundant requirements are included, but in case that requirements are changed it would increase the effort since every redundant requirement also needs to be changed. With redundancy the maintainability would be decreased.

Another class that we address together with redundancy is similarity. This means that requirements address the same issue, but the contents are not subsumed. For consistency, CRYSTAL's RBE therefore identifies 2 classes which will be described and classified in more detail in the following:

- consistency by no conflicts
  - on structural level
  - on logical level
  - on logical level including the environment and user knowledge
  - regarding ranges
  - regarding measurement units
  - with respect to the order
  - with respect to the timing constraints
  - between levels of detail
- consistency by reducing redundancy or similarity
  - combination
  - aggregation or entailment
  - necessity

It's important to note that conflicts can occur in several ways.

Assuming an underlying meta-model (e.g. an ontology described by logical description or any other means), structural inconsistencies might occur. For instance, having a concept for requirements and a concept for perspectives as well as a "*has-role*" between them, the demand of having all requirements in a *has-relationship* with an instance of perspectives and the fact of having a non-linked instance within the requirements results in an inconsistency.

For inconsistency on the logical level, we could assume a description of the requirements in some logic. For instance, the implications

$$a \rightarrow b \text{ and } b \rightarrow \neg a$$

for any events *a* and *b* are obviously inconsistent. Example requirements would be “If the car decelerates with at least 23 km per hour then the airbag opens” and “If the airbag opens, the car does not decelerate with at least 23 km per hour”, where event a is “the car decelerates with at least 23 km per hour” and event b “the airbag opens”.

The consistency on logical level including the environment and user knowledge is of interest since an inconsistency on purely logical level has few values if e.g. some events never occur. The information about the environment can lead to less or more inconsistencies.

Taking the example from above, there would be no inconsistency if *a* (i.e., the car decelerates with at least 23 km per hour) never occurs. This is clear, since

$$(a \rightarrow b) \wedge (b \rightarrow \neg a)$$

is equivalent to

$$\neg a.$$

On the other hand, there can arise even more conflicts by taking into account the environmental assumptions. For instance, we have the requirements

$$a \rightarrow b \text{ and } c \rightarrow \neg b$$

and the environment states that *a and c* hold. For instance, we have the requirements “If the car decelerates with at least 23 km per hour then the airbag opens” and “If the car drives then the airbag does not open” and the knowledge on the environment that “the car decelerates with at least 23 km per hour” and “the car drives” can occur at the same time, where the events *a and b* would be as before and event *c* is “the car drives”.

Inconsistency regarding ranges can appear if the events are said to happen in two not matching ranges (e.g. intervals). For instance, the two requirements “If the motor is running then the temperature of the air-conditioning is never below 35 Celsius” and “If the motor is running then the temperature of the motor is always below 30 Celsius” are inconsistent regarding the ranges.

For measurement units’ consistency, and following with the example above, it’s not a good practice to mix Celsius and Fahrenheit in the same specification.

The following table summarizes the metrics for Consistency that the brick partners would like to support at the end of the project. The purpose of the bricks is that the industrial partners will produce a ranked list of the metrics in their importance order. The bricks, depending on technical and skills, should try to fulfil the order.

Table 4-2. Metrics for Consistency

CONSISTENCY (C2)
A- Based on Requirement-Requirement consistency
Inconsistent information viewpoint
C2-A1-1- Is every requirement NOT conflicting (not contradicting) with any other in the studied specification?
C2-A1-2-Is every requirement coherent (in regards to complexity decomposition) with all requirements in the higher level trace line?
Inconsistent Units viewpoint
C2-A2-3-Is every requirement coherent (in regards to measurement units) with all requirements in the analyzed specification?

Redundancy viewpoint
C2-A3-Is every requirement NOT similar to a determined level to any other requirement in the selected specification?
Combination viewpoint
C2-A4- Are a set of requirements possible to be combined in one requirement?
B- Based on Requirement-SKB consistency
C2-B1-1-Is every requirement NOT conflicting (not contradicting) with any CRAC (Condition, Restriction, Assumption, Constraint) from the SKB?
C2-B1-2-Is every requirement NOT conflicting (not contradicting) with any sub-Artifact from the SKB?
C- Based on Requirement-Artifact consistency
C2-C1-1-Is every requirement NOT conflicting (not contradicting) with all modeling elements linked to the requirement by traceability?

4.1.3 Completeness

Completeness is a very difficult point to achieve, in other words, incompleteness is very difficult to detect. To formally reason about completeness, one needs a very precise definition of its meaning. Nobody and no method can assure for instance whether all failure modes of a system are identified if a complete list of them has not been compiled in beforehand; there could always be a forgotten mode. Moreover, human input is often required, e.g. the identification of all environmental influences of system (such as temperature and pressure) can be extracted from a text but a human is usually also necessary to decide whether there might be more - not mentioned - environmental conditions.

CRYSTAL's RBE approach will differ between the following notions of completeness:

- structural completeness
  - existential structural completeness of a concept
  - existential structural completeness of a role from one concept to another
  - universal structural completeness of a concept
  - universal structural completeness of a role between two concepts
  - Existential completeness based on viewpoints (functionality, performance, etc.)
- completeness with regard to content
- completeness with regard to the order
- range completeness

Structural Completeness refers to a meta-model (e.g. an ontology) of the system. Usually the design of a system, and its meta-model refers to different abstraction levels and different perspectives (such as operational, functional, logical or technical). Via for e.g. a meta-model represented as an UML-class diagram which reflects the structure or architecture of a system in a certain abstraction levels and a certain perspectives, elements of the meta-model are identified and characterized. Existential structural completeness of a concept is the question whether the concepts defined by a meta-model do indeed have instances.

Existential structural completeness of a relation from a concept C1 to a concept C2 asks whether all instances of C1 are related to C2 (i.e. any instance of the latter concept).

Universal structural completeness of a concept is the question whether all relevant instances of a concept have been defined. Unfortunately "relevant" is a very vague word. If there is a list of relevant instances in

some guide, one could of course check them, otherwise a human has to check the completeness. But still one might never know whether real completeness is achieved. Universal structural completeness of a role asks for the completeness of a role. This means, if a role can be established between two concepts (i.e. for two instances, one of each concept) there might be this role-relationship between them or not, then the question is whether all those relations have been identified and stated.

Completeness with regard to the content is self-explanatory. On the one hand it is a criterion where human involvement might be needed. On the other hand, if requirements are formalized (e.g. via a logic), the check whether requirements entail other requirements can be automated. When there is a list or some description which aspects should be addressed, it might also possible to automatically check for this point.

Range completeness asks for the completeness of ranges of variables. It is usually settled in the technical perspective and in the meta-model as specification.

The following table summarizes the metrics for Completeness that the brick partners would like to support at the end of the project. The purpose of the bricks is that the industrial partners will produce a ranked list of the metrics in their importance order. The bricks, depending on technical and skills, should try to fulfil the order.

Table 4-3. Metrics for Completeness

COMPLETENESS (C3)
A- Based on Viewpoints
Functional Behavior viewpoint
--- Prohibited behavior
C3-1-Are all prohibited behavior characteristics explicitly stated?
C3-2-For a required behavior, should there be an associated prohibited behavior defined, and if yes, is the prohibited behavior defined?
C3-3-Are requirements defined for all identified prohibited behaviors of the system?
--- Operational Scenario Identified
C3-4-Are all Operational phases covered?
C3-5-Are requirements defined for all identified Operational phases of the system?
C3-6-Are all Maintenance phases covered?
C3-7-Are requirements defined for all identified Maintenance phases of the system?
Architectural viewpoint(including internal interfaces)
C3-8-Are all the functional requirements fully allocated to architecture or components?
C3-9-Are all kinds of non-functional software requirements found in the software requirements specification for a system component? Reformulated from (All software requirements shall be identified?
C3-10-Are all internal interfaces declared in the requirements specification?
C3-11-Are all functional interfaces declared in the requirements specification?
C3-12-Are all Hardware/Software interfaces declared in the requirements specification?
Design viewpoint
C3-13-Are all the design constraints defined, substantiated and addressed?
C3-14-Are all the design Assumptions defined, substantiated and addressed?
C3-15-Are requirements defined for all identified design constraints and assumptions of the system?
C3-16-Are all the derived requirements defined, substantiated and addressed?

Non Functional viewpoint

C3-17-Are System range restrictions adequately defined, substantiated and addressed?

C3-18-Are requirements defined for all interval limits of all the system range restrictions that must be applied to the system development?

--- Safety viewpoint

C3-19-Are all functions, hazards and failure conditions covered?

C3-20-Are safety constraints and assumptions adequately defined, substantiated and addressed?

C3-21-Are all safety risks assessed and reduced?

C3-22-Are requirements defined for all identified functions, hazards, failure conditions and safety risks of the system?

--- Performance viewpoint

C3-23-Are non-functional performance properties adequately defined, substantiated and addressed?

C3-24-Are requirements defined for all performance properties of the system?

External Interfaces viewpoint (maps the Based on Stakeholders needs)

C3-25-Are External systems adequately defined, substantiated and addressed?

C3-26-Are requirements defined for all external systems mentioned in the SKB?

C3-27-Are affected people adequately defined, substantiated and addressed?

C3-27-Are requirements defined for all affected people mentioned in the SKB?

C3-28-Are external processes adequately defined, substantiated and addressed?

C3-29-Are requirements defined for all external processes mentioned in the SKB?

Environmental viewpoint

C3-30-Are environmental constraints and assumptions adequately defined, substantiated and addressed?

C3-31-Are requirements defined for all identified environmental constraints and assumptions of the system?

C3-32-Are Physical environment range restrictions adequately defined, substantiated and addressed?

C3-33-Are requirements defined for all interval limits of all the physical environment range restrictions that must be applied to the system development?

Production, Operation and Service viewpoint

C3-34-Are requirements regarding production defined at system, Hardware sub-system and software subsystem levels in the specification?

C3-35-Are requirements regarding service (repair and maintenance) defined at system, Hardware sub-system and software subsystem levels in the specification?

C3-36-Are requirements regarding decommissioning defined at system, Hardware sub-system and software subsystem levels in the specification?

C3-37-Are Production concepts adequately defined, substantiated and addressed?

C3-38-Are requirements defined for all identified production concepts in the Specification?

C3-39-Are service (repair and maintenance) concepts adequately defined, substantiated and addressed?

C3-40-Are requirements defined for all identified service concepts in the Specification?

C3-41-Are decommissioning concepts adequately defined, substantiated and addressed?

C3-42-Are requirements defined for all identified decommissioning concepts in the Specification?

Process conformance viewpoint

C3-43-Are Industry and company standards constraints and assumptions adequately defined, substantiated and addressed?

C3-44-Are requirements defined for all identified conformance rules, type planes, laws, regulations and standards that must be applied to the system development?

B- Based on Traceability

C3-45-Are all requirements in a specification linked? Reformulated from Requirements are linked to at least one lower level requirement?

Based on higher level requirements

--- Requirements cover completely upper level requirements

C3-46-Are all intermediate/low level requirements in the specification traced to upper requirements?

C3-47-Are all System functional, performance and safety-related requirements allocated to Software traced from the software requirements of the specification?
C3-48-Are all upper requirements covered by at least one lower level requirement?
C3-49-Are all functional requirements derived or refined by one or more sub-requirements?
C3-50-Are all safety requirements allocated to software components? (if Software components exists)
Based on Traceability to Sources
--- All sources of requirements are identified
C3-51-Are all the requirements of a particular specification traced to an identified source?
C3-52-Are all the Requirements Patterns defined for a particular specification being matched by the requirements?

Completely independent on views
C3-53-Are all requirements in a specification avoiding the TBD + TBC expressions?

4.2 Means to address CCC

4.2.1 Generic Metric to cope with Correctness

4.2.1.1 Correctness based on textual content (CORR1)

These metrics are already provided in RQS current version and are addressing by first indexing the requirement and then checking for the use of the proper concepts (or, in other concepts, checking the absence of improper concepts), the proper verb tense, mode… Finally, readability criteria is also included. [CRYSTAL Deliverable: CRYSTAL_D_607_021]

4.2.1.2 Correctness based on structure (CORR2)

After a semantic indexing of the requirement, and by finding a corresponding matching between the requirement and the suitable set of patterns, every requirement will be deemed as correct or incorrect according to the matching against, at least, one suitable pattern.

4.2.1.3 Correctness based on formal representation (CORR3)

Based on the standard formalization, as well as the additional transformation described as T4 in section 2.1.3.3 , any further correctness metric, customized and developed by the end-user, will be available in RQA and RAT. To support that, RQS will integrate mechanisms to extend the list of correctness metrics by providing the suitable set of libraries.

An interface for those libraries, as well as examples of use will be provided in this Workpackage.

4.2.1.4 Correctness based on Management elements (CORR4)

These metrics will not require any complex NLP system nor the support of ontology. Instead of that, some elements such as the presence of absence of links, the level in a requirements hierarchy, the fulfilment of a specific set of mandatory attributes… will be checked.

4.2.2 Generic Metrics to cope with Consistency

Several metrics can be understood to be part of the consistency section. A first way to organize them could be with regards to the comparison agent. The following main agents are foreseen:

- Consistency of a requirements specification against the System knowledge Base (Ontology)
- Consistency of a requirements specification against other requirements specification
- Consistency of a requirements specification against other systems engineering artifact/model

4.2.2.1 Consistency against ontology

4.2.2.1.1 Consistency against Restrictions (CONS1)

The main objective of this technique will be to measure consistency of a requirements specification based on the idea that a requirement's restriction must not conflict with any restriction in the ontology. This metric has its counterpart in the correctness metric based on restrictions. The main idea in this metric is that instead of checking that the restrictions defined in the ontology must be found in the specification (the correctness metric), in this metric the operation is to see that the restrictions defined in the ontology do not conflict with the restrictions defined in the requirements specification. A clear definition of what a restriction is (at conceptual level) will be needed

For example:

If the following Restriction is defined in the ontology:

Function A must be performed after Function B

and the specification says:

The Function A shall receive its input from Function B

It could be possible to state that the specification is INCONSISTENT, as there is one restriction in the ontology that is not fulfilled by the requirement.

4.2.2.1.2 Consistency against System Conceptual Model (CONS2)

The architectural consistency analysis can also verify the consistency of requirements w.r.t. to the system structure (SKB). This metric can check if a requirement refers to artifacts of the SKB correctly. For example if a requirement implies or relies on some structure of the SKB these facts have to be represented in the same way within in the SKB.

4.2.2.2 Consistency of two requirements specifications

4.2.2.2.1 Consistency of two specifications based on their restrictions (CONS3)

The main objective of this technique will be to measure consistency of a requirements specification based on the idea that a requirement's restriction do not conflict with other requirement's restrictions. This metric has its counterpart in the correctness metric based on restrictions. The main idea in this metric is that instead of checking that the restrictions defined in the ontology can be found in the specification (the correctness metric), in this metric the operation is to see that restrictions do not conflict with each other in the same or in different requirements. As in completeness, the restrictions must be predefined in the ontology, at the corresponding level.

For example:

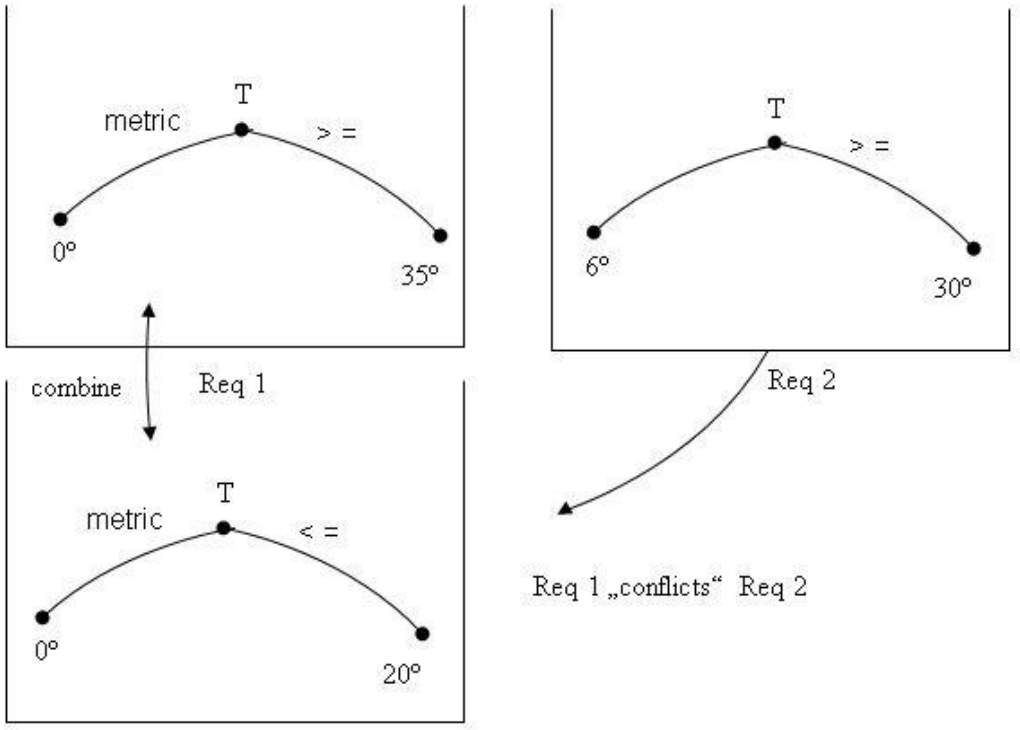


Figure 4-1. Consistency based on restrictions

The way to solve this metric is by predefining the operations that must be performed for every restriction to analyze if it conflicts with other restriction of the same type.

In the example above:

Req1 restriction states that Temperature >= 35ºc

Req2 restriction states that Temperature >= 30ºc

The way to analyse a range restriction (in case of restricting the same element) is by comparing that the intervals are consistent.

Figure 4-2. Consistent intervals

Based on algebraic calculations.

4.2.2.2.2 Consistency of two Specifications analyzing Higher-Lower abstraction level (CONS4)

The main objective of this technique will be to measure consistency of two requirements specification based on comparing abstraction levels of how high level requirements are derived into low level requirements.

The following metrics should be considered:

- Compare COMP1 values (see generic metrics for completeness) for High level specification with COMP1 values for Low Level Specification.
  A predefined threshold should be determined for the comparison calculation. If the calculated results are higher than a threshold, then the metric is not fulfilled.

- Smart comparison of hierarchies. Given the hierarchies formed by the existing terms in a Specification found in the System Thesaurus, perform coverage analysis with them.
  For example:

  If Specification1 (SP1) is the High level specification and has a coverage in a PBS drawn as red and Specification2 (SP2) is the low level specification and has a coverage in a PBS drawn as blue, the following drawing shows a problem because the high level specification has deeper concepts than the low level, which has higher concepts that the higher.

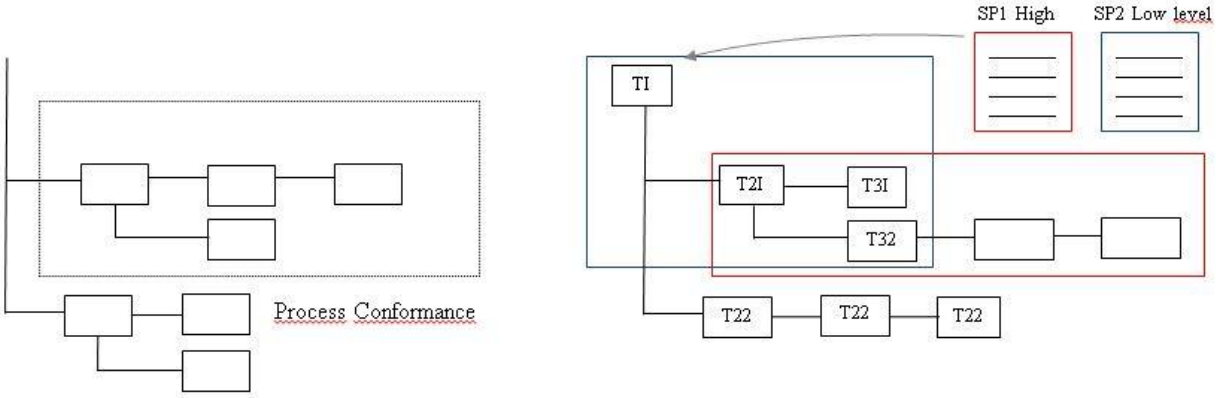


Figure 4-3. Consistency based on levels of abstraction

The application of mean and variance calculations for the terms and their levels in the ontology will be used, comparing both calculations between the two specifications, as well as against the ontology.

4.2.2.2.3 Entailment/consistency (CONS5)

During the development process, the requirements of components are further refined or even new requirements may be added. A consistency analysis can support the requirements engineer to find inconsistencies within the requirements early in the process. After changing an existing or adding a new requirement, the analysis offers an efficient way to check if the new set of requirements includes any contradictions. This analysis includes all assumptions and promises of the contracts as well as the relation between the assumptions and promises of each individual contract.

More precisely, the analysis checks the existence an execution trace which fulfills all requirements. Therefore, the requirements are translated into corresponding Linear Temporal Logic (LTL) formulas and an LTL model checking procedure is applied.

The entailment analysis supports the engineer during the decomposition of a system into multiple subsystems or after a (sub-)system requirement has been changed. It performs a virtual integration test (VIT) which is only based on the contracts.

The main idea is that the combined contracts of the sub-components must fulfill (imply) the contracts on the parent component. The entailment analysis can verify this relation for all parent contracts and all subcontracts (or for a selected subset) by exploiting the formal semantics of the RSL.

Both types of analysis should also work with probabilistic requirements (e.g. requirements expressing the failure/error rates of components).

4.2.2.2.4 Smart Consistency Analysis (CONS6)

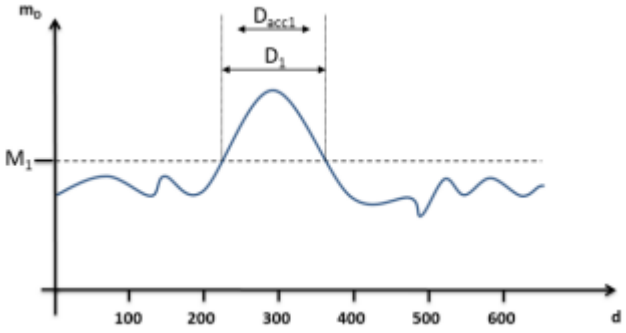
The main idea of the smart consistency analysis is to further optimize the formal consistency analysis procedures. A formal consistency verification of a large set of requirements is often not possible due to limitations in the size of the generated formal models for the model checking engines. Therefore, the smart consistency analysis tries to split the set of requirements in question into smaller subsets which are potentially inconsistent. The normal consistency analysis is performed afterwards on these sets and a verdict for the complete set of requirements can be inferred.

4.2.2.2.5 Consistency regarding the ordering/timing (CONS7)

OFFIS offers two tools that deal with this criterion. They rely on properties that denote sets of traces. A trace is a valuation of a variable over time, formally:

1. $\mathcal{T}$ *is the set of* points in time, $\mathcal{V}$ *is the domain of values the variables may take.*
2. *The set of traces* $\mathcal{S}$ *is a subset of* $[\mathcal{X} \rightarrow [\mathcal{T} \rightarrow \mathcal{V}]]$.

An example trace is shown as blue line in the following graphics:



A requirement that can be analyzed might look like: “The moment M does not exceed M_1 for more than D_{acc1} ms.”. The above trace violates this requirement. A set of requirements is inconsistent iff there exists no trace satisfying all of them.

Sometimes not the actual time points are relevant, but just the ordering of certain events. Therefore there are requirements that state directly or indirectly any ordering, e.g.,

- If event A occurs, then event B is triggered,.
- Directly after the event A, always the event B occurs.
- At some time after event A, the event B will occur.

- The consistency checker can be used to discover conflicts
  The input is a set of requirements and it outputs “yes” if the set of requirements is consistent and “no” otherwise.
- The entailment checker can check consistency between higher and lower level requirements and aggregation of requirements regarding the ordering.
  The input are a single requirement (e.g. a high level requirement) and a set of requirements (e.g. a set of lower level requirements). The entailment checker outputs “yes” if the the single requirement entails the set of requirement and “no” otherwise.

How could the checker be integrated into the REUSE tools:

1) The patterns that can checked via the consistency or entailment checker must have a special structure and RSHP. Therefore when they are created, they are allocated to an inference rules group called “checkable patterns”. Adding checkable patterns requires consulting from OFFIS/REUSE
2) For the start-up of the checker, active entries are necessary since the set of requirements that are considered must be chosen. Therefore the tests can be selected by a button in the RQA. Then the user must decide whether she wants to check for consistency or for entailment.
   a. If the user selected consistency, a set of requirements must be chosen.
   b. If the entailment checker was selected, first a module and a requirement are selected for the origin of the entailment. Secondly a module and a set of requirement is selected that should be entailed by the single requirement.
   c. (TBD) based on pre-analysis requirements (as inputs for theses analyses) might be selected automatically
3) The output of the tests is “yes” or “no”. In the extended version the set of inconsistent requirements will be narrowed automatically.

Example

1. Assumption: Only pressure of the brake pedal is a user interfering for braking.
2. Whenever the pedal of the brake is pressed, the car will decelerate immediately.
3. If an accident happens, then the car shall decelerate within 10 ms.
4. The car shall never decelerate without user-interfering without

Obviously, the third requirement is not consistent with the second.

4.2.2.3 Consistency of a Specification against other artifacts

4.2.2.3.1 Consistency against architectural models (CONS8)

The architectural consistency analysis can also verify the consistency of requirements w.r.t. to the system structure (SKB). This metric can check if a requirement refers to artifacts of the SKB correctly. For example if a requirement implies or relies on some structure of the SKB these facts have to be represented in the same way within in the SKB.

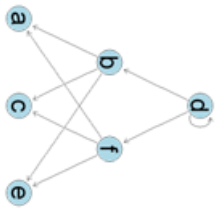
4.2.2.3.2 Consistency of a Specification based on transformations (CONS9)

The principles of this metric should be found on a generic transformation approach. The idea behind this metric relies on the principle that in order to check if two requirements conflict (or do not conflict), a formal model for the requirements is needed. The possibility to apply different transformation operations to the model in one requirement trying to reach the other model representation could allow to make assumptions about possible consistence or inconsistence.

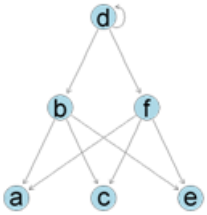
In order to develop this metric, a domain specific language shall be defined and implemented, where specific operations should be offered to the Domain Architects to define possible transformations.

For example:

If a requirement has a formal representation F1

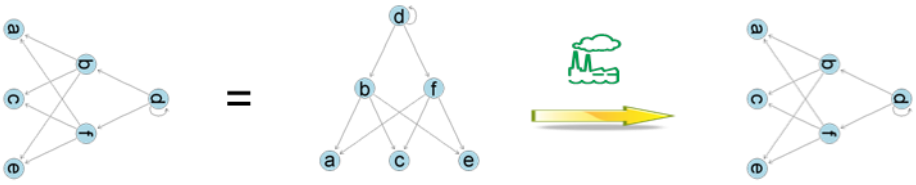


And if requirement 2 has a formal representation F2



And if a transformation T1 is defined for accepting consistency in the way that, if it is possible to reach F1 applying T2 to F2, then the requirements are consistent.

We could assure that requirement 1 and requirement 2 are consistent when the transformation succeed.



4.2.3 Generic metrics to cope with Completeness

Completeness of a specification should measure how complete a given specification made of requirements (written in natural language text) is.

Completeness is a very difficult measure, as it is clearly dependent on what the definition of “the whole” is. Without the delimitation of “the whole” it is not possible to calculate completeness values.

On the other hand, even if the definition of “the whole” exists, it cannot be assured that “the whole” includes all the knowledge to be defined about a system. Therefore, the completeness measurements will inform you about incomplete requirements, but won’t be able to state that requirements are complete.

However, assuming that a SKB (System Knowledge Base) exists, even if it will never formally represent in a complete way a system, it should be used as the model for “the whole”, to analyze completeness. Therefore, completeness metrics could be defined to check how the information extracted from requirements covers all the different layers of the ontology.

4.2.3.1 Completeness of a Specification based on Terminology coverage (COMP1)

The main objective of this technique will be to measure completeness of a requirements specification based on the terms it contains in comparison with the terms existing in the vocabulary layer of the ontology (more specifically, those belonging to a domain ontology).

Considering:

- TONS = List of Terms Existing in the Selected Branch of the Ontology and NOT used in the Specification
- NTNO = Number of Terms in Specification Not existing in the Ontology
- NTEO = Number of Terms In Specification Existing in the Ontology
- NTONS = Number of TONS
- NTS = Total Number of Terms in the Specification = NTNO + NTEO
- NTO = Total Number of Terms in:
  - Selected Branch of the Ontology (or the whole ontology)
  - Selected Semantic
- RNO = List of Requirements with at least one of its Terms NOT included in the Ontology
- RO = List of Requirements with all its Terms included in the Ontology
- NRNO = Number of RNO
- NRO = Number of RO

The following measures shall be calculated:

- Overall Completeness coverage rate 1 : NTNO / NTS
- Overall Completeness coverage rate 2 : NTNO / NTO
- Completeness coverage rate 3 : NTEO / NTS
- Completeness coverage rate 4: NTEO / NTO
- Completeness coverage rate 5: NTS / NTO
- Completeness coverage rate 6: NRO / (NRO + NRNO)
- Completeness coverage rate 7: NRNO / (NRO + NRNO)
- Completeness coverage rate 8: NTONS / NTO

Other information shall be included in the reports:

- RNO
- RO
- TONS

- Requirements by Term
- Term Frequency: classified in different clusters:
  - the concepts in the domain thesaurus
  - the concepts from the forbidden lists (ambiguous and other lists of forbidden concepts for correctness). To be used to better understand correctness issues in the requirements
  - the terms not existing in the ontology. To be used as possible suggestions to extend the business ontology

A threshold limit shall be defined for every value as part of the metric configuration. If the actual value goes beyond or below the threshold → the specification will not succeed in the metric.

For example:

> *If a whole requirements specification has 1000 terms and the selected branch in the ontology has 1100 terms,*
>
> *the Completeness coverage rate 5 = NTS / NTO = 1000/ 1100 = 0,91*
> *if the threshold limit defined is [0,88, 0,92] => this value is correct for passing the completeness metric (coverage rate 5)*

4.2.3.2 Completeness of a Specification based on Structure coverage (COMP2)

The main objective of this technique is to measure completeness of a requirements specification based on the fundamentals that the requirements specification includes (as a whole) a fixed and previously determined group of structural breakdown hierarchies. The hierarchies must be predefined in the ontology, using whole-part relationships between terms. Completeness shall be measured by acknowledging that all the hierarchies selected from the SKB (Ontology) are found in it the requirements specification.

This metric aspires to calculate which percentage of the terminology (hierarchical whole-part relationships) exist in the specification.

In order to configure this metric, it would be necessary to define, in one side a section of the SKB containing breakdown structures that will be controlled, and in the other side a Requirements Specification (module, project).

4.2.3.3 Completeness of a Specification based on Functionality coverage (COMP3)

Similar to COMP2, but functionality coverage seems to be a more complicated problem than structural decomposition.

It is known in Systems Engineering that functional architectures have other relationships than decomposition (for instance Functional data flow, or control flow, etc..) and therefore, functional requirements specifications can be affected by these problems. Additional discussion between industrial partners and UC3M and REUSE will be needed.

4.2.3.4 Completeness of a Specification based on Interface definition coverage (COMP4)

The information about the structure of the system (SKB) and its interface definitions can be used in a completeness metric which checks if all specified data-flows (e.g. inputs of a subsystem) are covered with the requirements.

In addition, the coverage of the different domains (ranges) of these data-flows can be checked by completeness metric. This can check if the complete range of input values of a subsystems is covered by at

least one requirement. If parts of the domains are not covered, the requirement specification has to be considered incomplete.

4.2.3.5 Completeness of a Specification based on Properties coverage (COMP5)

The main objective of this technique is to measure completeness of a requirements specification based on the fulfillment of a predefined set of properties (not functional properties).

The properties must be predefined in the ontology, at the corresponding level. Completeness shall be measured by acknowledging that all the properties to be applied to a specification are found in it.

A clear definition of what is a property (at conceptual level) will be needed although the concept refers to the so-called not functional “ilities” of every system (safety, performance, reliability etc..)

4.2.3.6 Completeness of a Specification based on Restrictions coverage (COMP6)

The main objective of this technique will be to measure completeness of a requirements specification based on the fulfillment of a predefined set of restrictions.

The restrictions must be predefined in the ontology, at the corresponding level. Completeness shall be measured by acknowledging that all the restrictions to be applied to a specification are found in it.

A clear definition of what a restriction is (at conceptual level) will be needed.

For example:

If the following Restriction is defined in the ontology:

Function A must be performed after Function B

and the specification says:

The Function B shall receive its input from Function A

It could be possible to state that the specification is not INCOMPLETE, as all the restrictions in the ontology are fulfilled in the specification.

One of the main aspects of this metric will be to define in the ontology the different restrictions types.

But in order to visualize/manage them we should consider at least two different ways to define and represent restrictions:

Restrictions described using a graphical model (SysML)

- An Activity diagram for representing sequences of activities
- A sequence diagram for representing sequential object communications
- A State chart diagram for representing States and Modes restrictions.
- A class diagram for representing structural connections

Restrictions described as textual elements in natural language

- The temperature must be between -12ºc and +15ºc

Completeness can happen at different levels:

- L1: Every requirement of a determined Pattern shall match a determined restriction or be consistent with a restriction model.
- L2: All the selected restrictions in the ontology must be matched inside a whole specification (module or project). If one restriction is not matched => the Completeness metric fails.

Every Level would imply the definition of a Completeness’s metric.

4.2.3.6.1 Different restriction types:

- Sequence Restriction: “Function 1” is performed after “function 2”

  A Completeness metric for sequence restrictions should be applied at level L2. It means that a set of sequence restrictions will be defined in the ontology and selected to occur in a Specification, and if one of them is not fulfilled → not successful.

  Example:
  The following Restrictions are defined in the ontology:
  - Function A must be performed after Function B
  - Function B must be performed after Function C

  If the specification says:
  - The Function B shall receive its input from Function A
  - The Function C shall receive its input from Function B
  - The Function A shall receive its input from Function D

  The specification is not incomplete regarding ordering.

- Ranges Restriction: “Component1” Operates in a defined range]or [XX, YY] or [

  A Completeness metric for Ranges Restrictions should be applied at level L2. That means that a set of Ranges restrictions will be defined in the ontology and selected to occur in a Specification, and if one of them is not fulfilled => not successful.

  Example:
  The following Restrictions are defined in the ontology:
  - Environment Air temperature restricted to [-40º C, +50º C[

  If the specification says:
  - The engine shall work at an environment Air temperature higher or equal than -40 ºC
  - The engine shall work at an environment Air temperature lower than +50 ºC

  The specification is complete

- Operational Restriction: fulfillment of a State machine

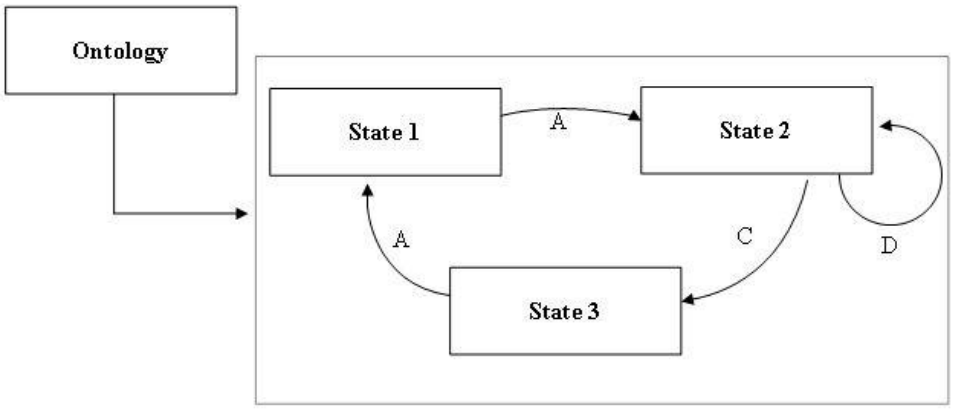


Figure 4-4. Fulfilment of a State Machine

- Collaboration restriction: collaboration diagram
  Etc..

4.2.3.7 Completeness of a Specification based on Patterns coverage (COMP7)

The main objective of this technique will be to measure completeness of a requirements specification based on the distribution of the requirements within a set of Patterns. The requirements will be grouped according to the pattern matched and the group/s the pattern belong to.

The following measures shall be calculated:

- Distribution of Requirements within the selected Patterns, as the number of requirements that match every Pattern, including those Patterns that do not fire any single requirement
- The same distribution but based on Patterns groups
- Variation of the distribution according to an average value. For that case, it will be necessary to support the possibility to insert an average % of requirements distribution in the patterns.

  Example:
  If the ontology has 2 Patterns:
  P1 => Safety requirement
  P2 => Functional requirement

  RQA shall support the possibility to define, that for a particular project, the standard distribution of requirements among the Patterns is:
  P1 => 35%
  P2 => 55%
  Not matching a P => 10%

  A metric will be generated to compare the actual distribution of requirements between Pattern1 and Pattern2, and if a threshold limit is overpassed → the specification WILL NOT pass the completeness metric. Possible variants for this metric:
  - Checking the number and not the distribution (%)
  - Automatic calculation of the distribution (%) based on previous projects
- Lists of requirements by Pattern and group
- Lists of the requirements that do NOT match any Pattern

4.2.3.8 Completeness based on links (COMP8)

Using an extension of the in/out – link metric provided so far in RQA V4, the user must be able to count a specific type of in or out link.

A specification is not complete whether:

- A requirement in the top of the abstraction level is not linked to any requirements in the following level of abstraction
- The opposite: Considering also links to other artifacts (not only requirements) as design elements decisions (where a new requirements could have arisen)
- Using other types of links (such as tests), a specification can be not complete if any of the requirements has no test cases traced

(Note: Use type of links mentioned in CESAR Requirement Meta-model (RMM), see [CESAR_RSL,2011]).

4.3 Preliminary Integration Approach into RQS Tools

In the vision of The REUSE Company, a requirements quality management problem always starts by defining the requirements type that the organization will intend to control. For example, it could be necessary for an organization to manage the quality of “performance requirements”. Even if it seems to be simple, the selection of the kind of requirements to formalize is not trivial. “In order to define what I want, I usually need to know what I do not want”. And this is a real problem. Because, for example, in many cases “performance requirements” are NOT “functional requirements”, but, in other cases, they certainly are.

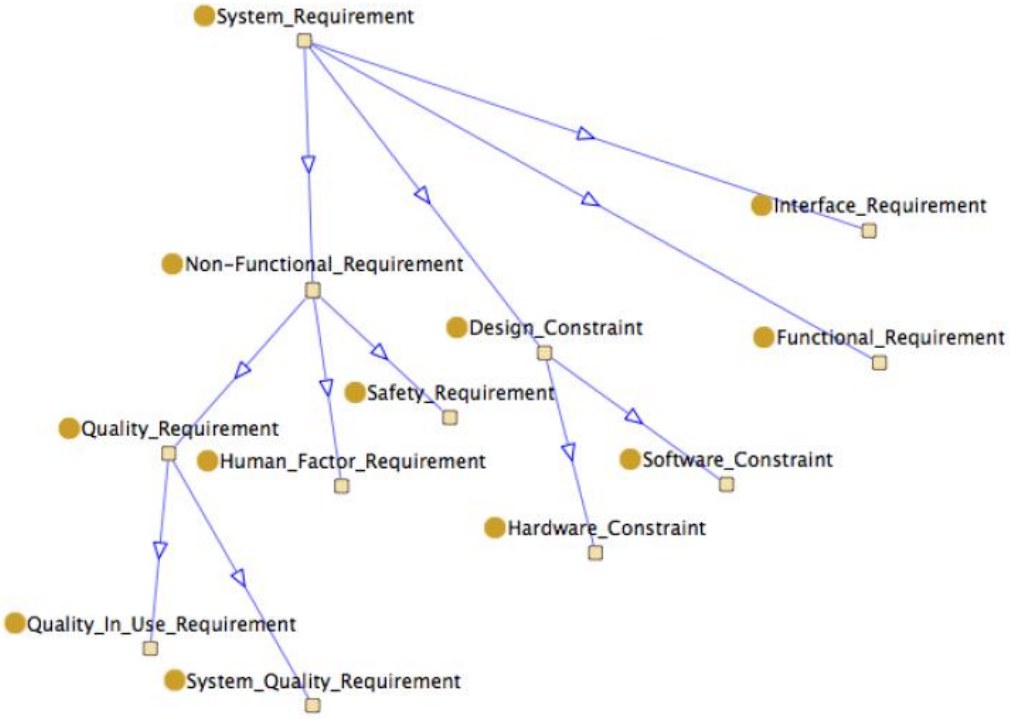


Figure 4-5. Taxonomy of types of requirements

However, this document will try to formalize one possible representation of a functional requirement: The requirements that define a functionality of a system when a condition is defined.

Some examples of requirements would be:

- If the user presses the brake pedal, the car shall reduce its speed immediately
- If the pedal of the brake is pressed, the car shall decelerate immediately
- Whenever the pedal of the brake is pressed, the car will decelerate immediately
- The call will decelerate immediately as soon as the pedal of the break is pressed

And so on:

In this chapter, we will present two extended examples. Our vision is that what is done by hand by us in these examples, should later be done automatically by tools on many different examples.

4.3.1 Example 1: Formalization possibilities in RQA

4.3.1.1 Problem formulation

We will use the following requirement as our running example:

"Whenever the pedal of the brake is pressed, the car will decelerate immediately"

We would like to arrive at the following representation:

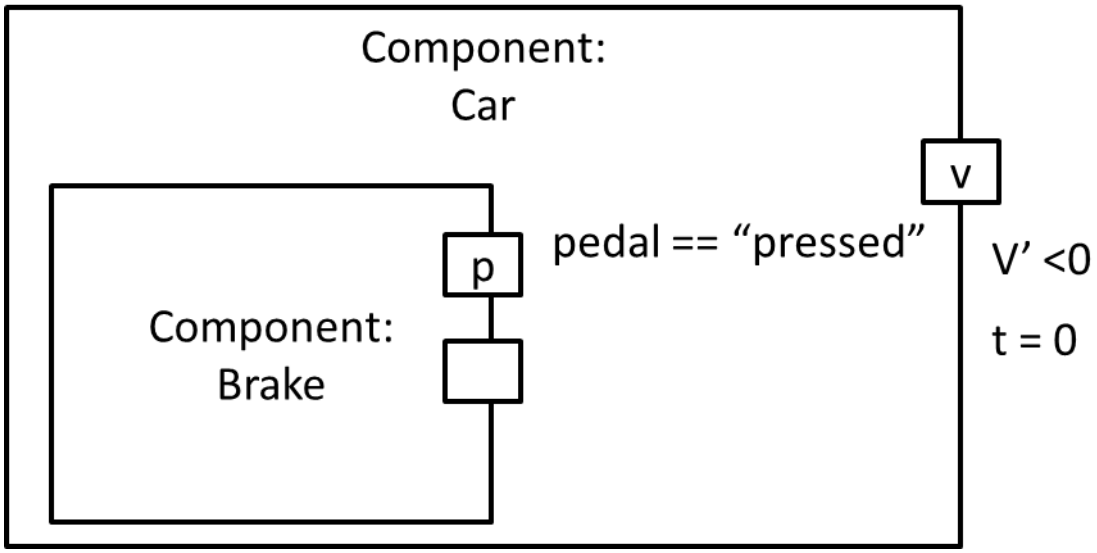


Figure 4-6: Formalized Requirement

This information is richer than the original requirement, as the requirement doesn't include all the different relationships stated in the previous model.

4.3.1.2 Vocabulary

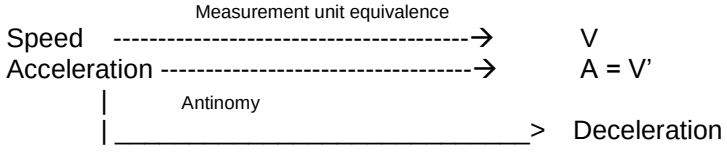
It is possible to extract terminology that must be in the vocabulary of the ontology. Here we have the following words: *Car*, *Pedal of the Brake*, *Decelerate*, *Immediately*.

4.3.1.3 System Thesaurus

The project breakdown structure (PBS) also contains useful information:

- Car = System
  - Brake = Brake System
    - Pedal of the brake
      - Pedal States: Pressed, Released

Knowledge about the physical world also help to understand the requirement:



4.3.1.4 Syntactic Patterns

A quick first syntactical analysis will present the following structures:

Whenever
The
Pedal of the brake
Is pressed
The
Car
Will
Decelerate
immediately

Two syntactic structures have to be considered here: The compound noun "*pedal of the brake"* and the compound verb "*is pressed".* The joining of those words as a single compound phrase is performed by the *Tokenization* stage of the process.

We´ll not produce patterns at syntactical level because we consider that "Pedal of the brake" is, in itself, a term of the domain with full meaning. This means that "Pedal of the brake" must be included in the ontology. The structure "is pressed" will be split in the two terms that form it: the verb "to be" and the verb "to press" in participle.

4.3.1.5 Requirements Engineering Patterns

Now the requirement is analysed based on its semantic parts:

The following functional organization will be used:

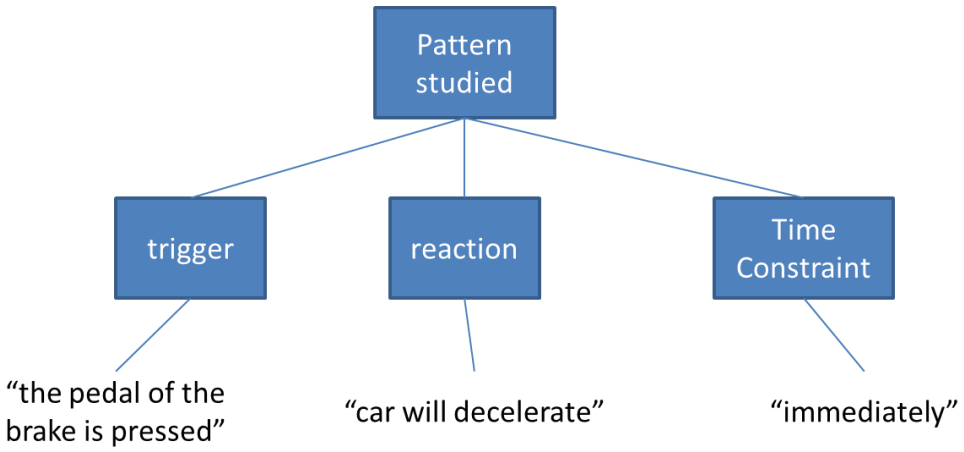


Figure 4-7: Functional organisation for requirement

Every functional part will form a pattern in the ontology.

The patterns will be:

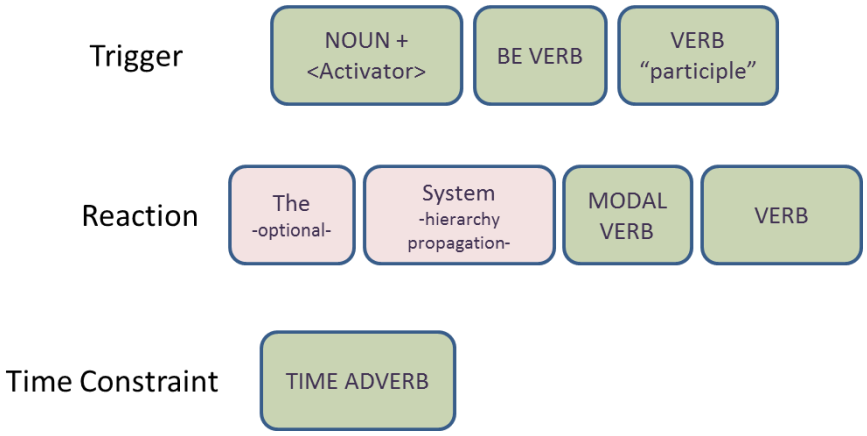


Figure 4-8. Example of pattern

The following formalization will be used:

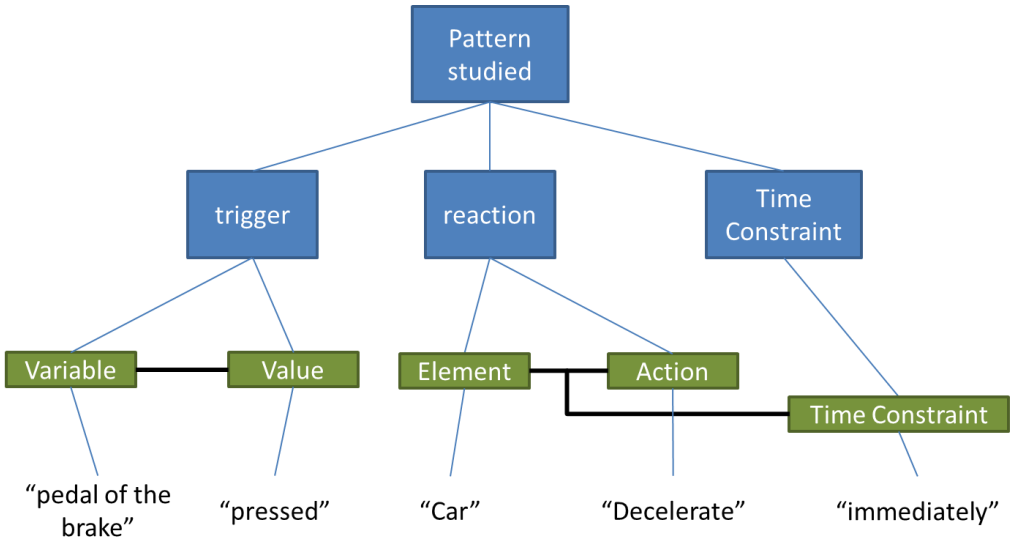


Figure 4-9: Formalization of the requirement

As it can be seen in the previous diagram, the trigger pattern produces by itself a graph:

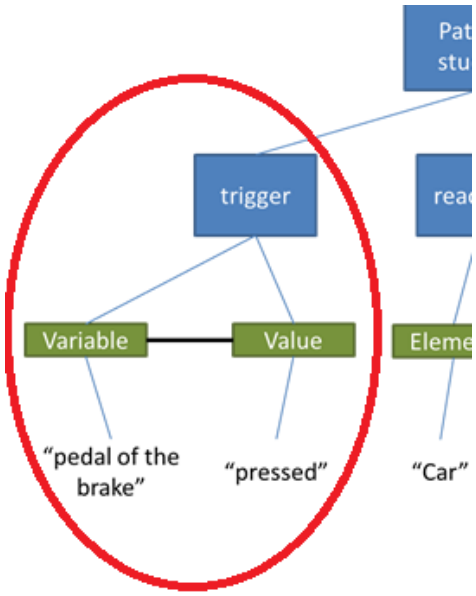


Figure 4-10: Graph of the trigger

The complete representation will be:

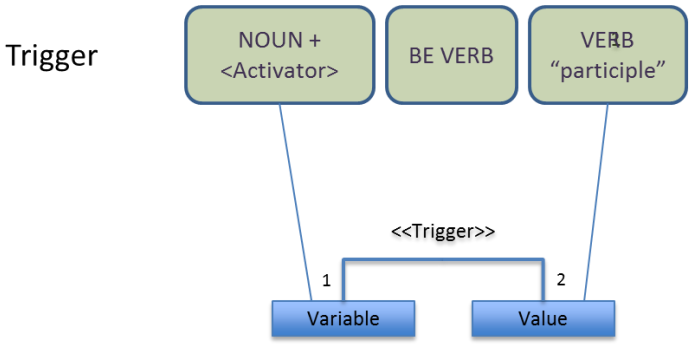


Figure 4-11. Example of formalization

But, a problem arises when trying to formalize the graphs out of the other patterns: the graph we want to produce is formed by elements coming from different patterns (reaction and time constraint):

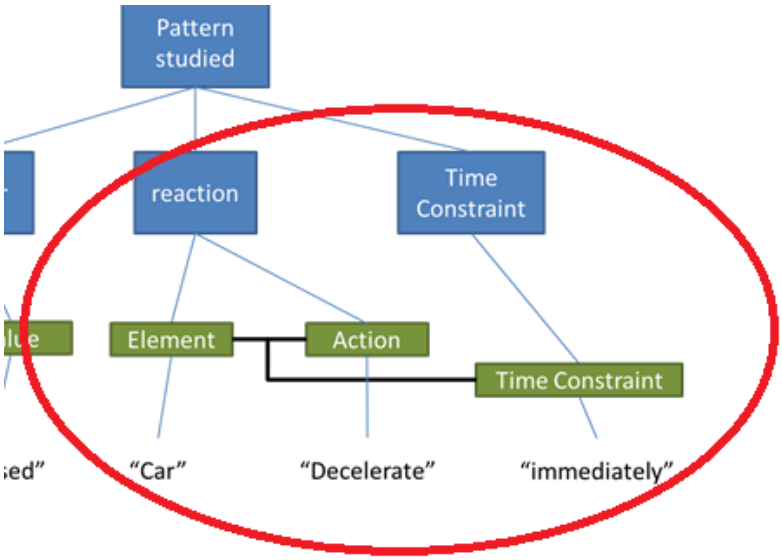


Figure 4-12. Example of formalization #2

This means that it is not possible to formalize the graph only at the Reaction pattern, and also not possible to formalize it at the time constraint pattern.

There are two solutions to this problem:

- Unify the reaction pattern to include a section for the time constraint.
- Create the graph at the next level, where both patterns exist as sub-patterns.

In our case, we´ll use the second solution, so the formalization will be produced at the next level of patterns. This formalization process is, of course, very human dependent, and we must not be afraid of that. The result of this process must be the way the organization understands the requirement semantics. Just by defining a way, good benefits of it can be gathered (sharing same understanding, promoting the view to the supply chain etc etc.)

For example, other Systems Engineering group could see the studied requirement as a state machine transition, and the modeling could be a different one.

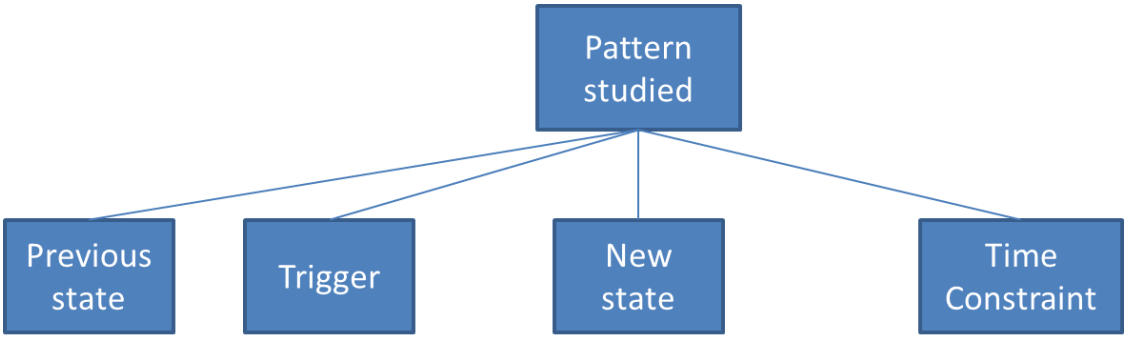


Figure 4-13. Pattern structure

Matching requirements of the type:

If the car is moving, whenever the pedal of the brake is pressed, the car will decelerate immediately

4.3.1.6 Classification layer of the ontology

The <Activation> Class was needed to for defining the Trigger Pattern, so it must be included in the ontology, assigning the term “Pedal of the brake” as instance.

On the other hand, the following Relationship types were created for the graphs production:

- <<Trigger>>
- <<Reaction>>

4.3.1.7 Requirements Patterns layer

One single pattern will be produced to map the requirement text:

Figure 4-14. Pattern layer

The defined sub-patterns can be found in the pattern, as well as the syntactical support elements to separate them.

As it has already been described, the formalization graph produced by this pattern will be the reaction graph, as it is affected by two sub-patterns.

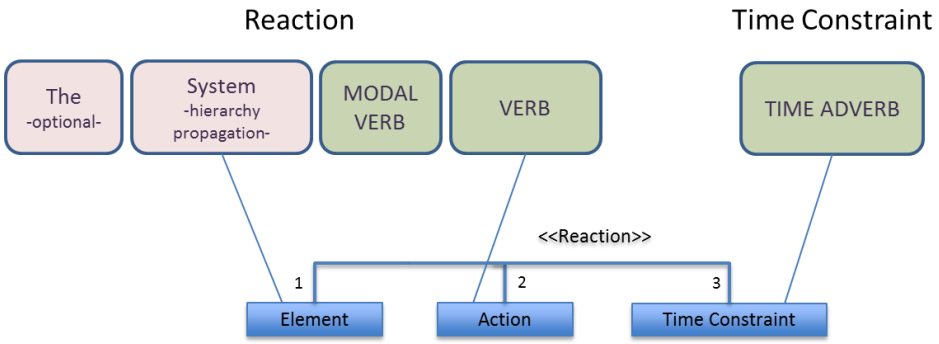


Figure 4-15. Formalization semantic graph

4.3.1.8 Ontology information: Summary

Here we summarize the following operations:

- The vocabulary and thesaurus layers have been populated with

Car = System
Brake = Brake System
Pedal of the brake ----------→ Pedal States
Pressed
Released

Physical Environment
Measurement unit equivalence
Speed ----------→ V
Acceleration ----------→ A = V'
| Antinomy
| _____> Deceleration

- The Light ontology layer includes

  - Classes
    <<Activator>>
    Pedal of the brake

  - Relationship types
    <<trigger>>
    <<Reaction>>

- The patterns layer includes

A set of NOT INDEXABLE internal patterns with their corresponding formalizations. The concept NOT INDEXABLE means patterns that cannot be instantiated by themselves, but that can be part of a larger patter (indexable or not, what permits the possibility of patterns, sub-patterns, sub-sub-patterns…):

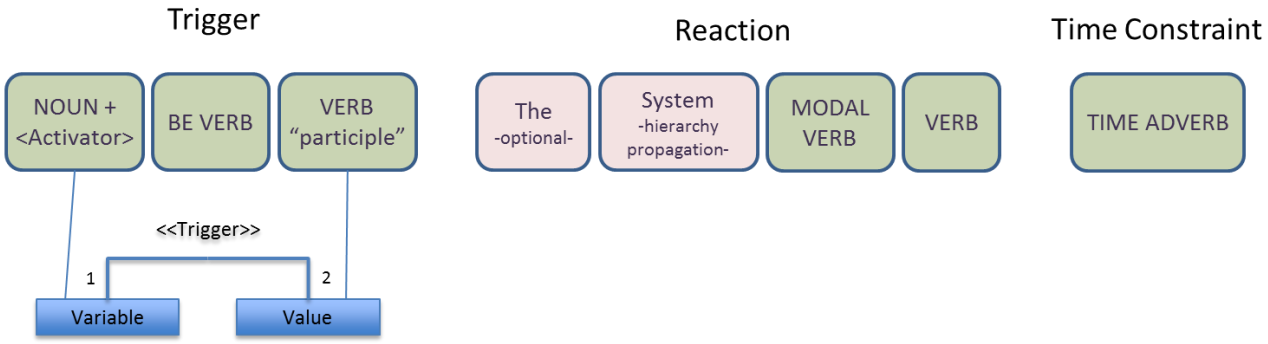


Figure 4-16. Not-INDEXABLE patterns

A set of INDEXABLE requirements patterns, with their corresponding formalizations:

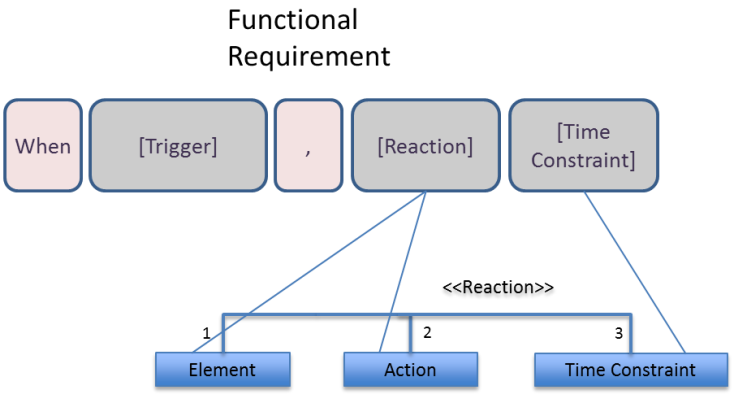


Figure 4-17. INDEXABLE pattern

Where, the graph is indeed produced by the corresponding internal elements of the sub-patterns:

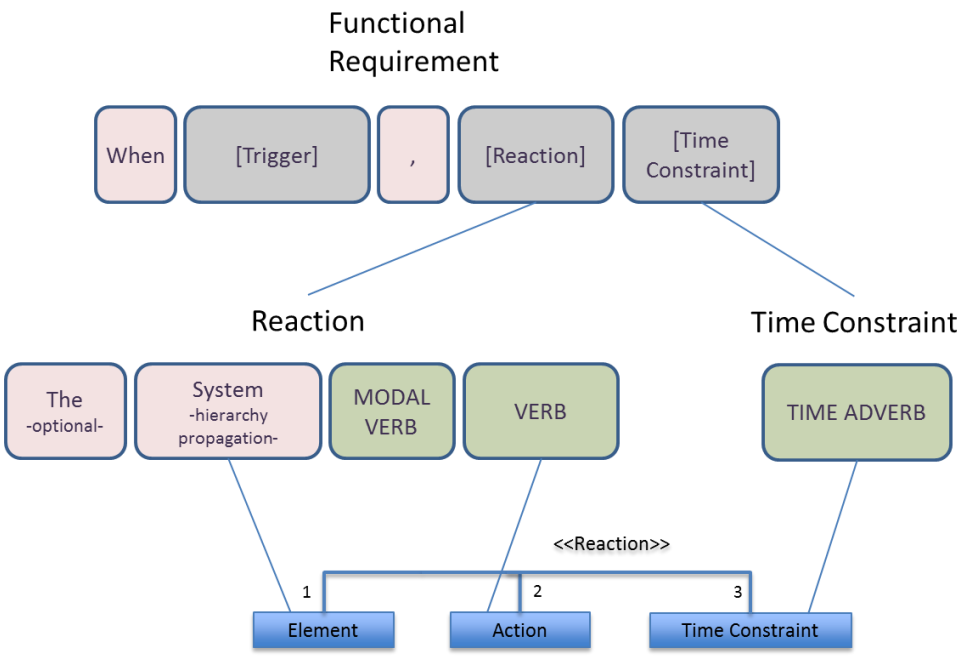


Figure 4-18. Formal representation

4.3.1.9 Formalization transformation and Post-processing

When the input requirement arrives:

Whenever the pedal of the brake is pressed, the car will decelerate immediately

RQA (as well as RAT - the Requirements Authoring Tool), will produce the following results after working with the formalization transformation (T2):

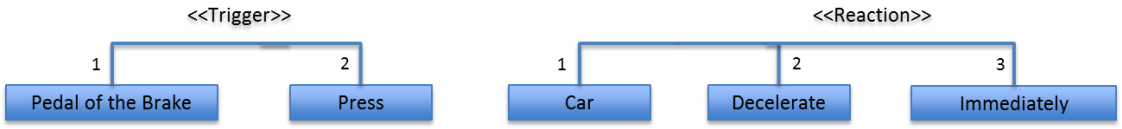


Figure 4-19. Indexing results in RQS

In order to get the expected results defined above a specific transformation must be produced to the graphs created so far (using the post- processing transformation). In order to represent:

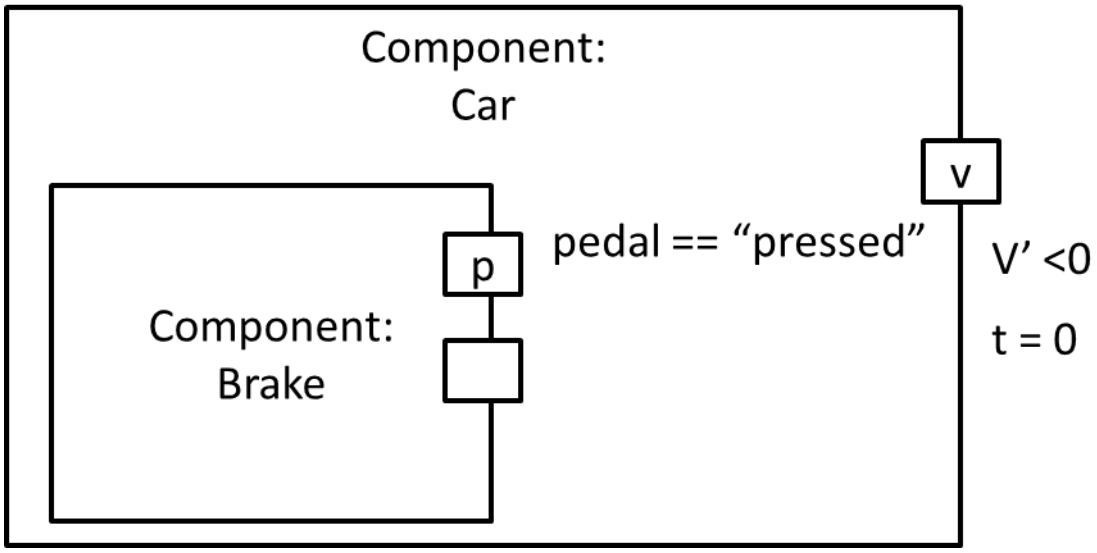


Figure 4-20. System representation

The graph should present the following structure:

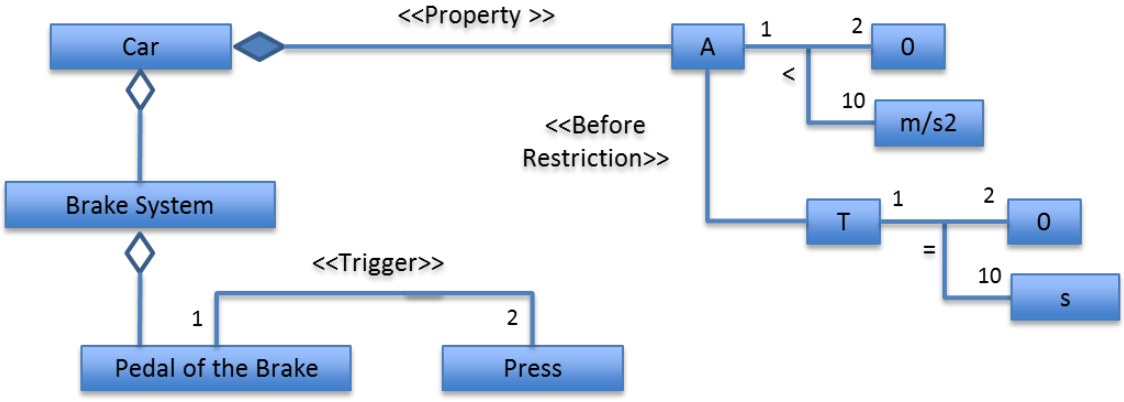


Figure 4-21. Complete formalization

Some slight differences can be found between both representations:

- The Pedal of the brake is modeled as a breakdown (an aggregation) instead of as a Property (as suggested)
- The requirement is not modeling that the car has a property named V (Speed) as it is NOT mentioned in the requirement.

And the input graph to be transformed is:

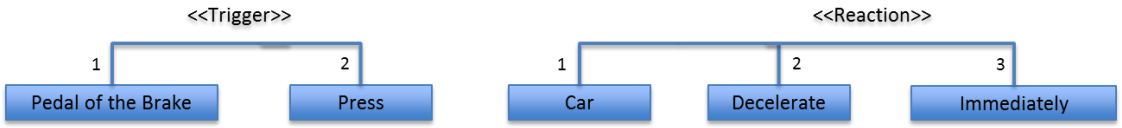


Figure 4-22. Transforming formal representations

So the transformation logic could be:

1- Look in the ontology (in the hierarchical structure) if the *"Element"* of the **[Reaction]** pattern can be found in the parent hierarchical structure for the "*Variable"* of the **[Trigger]** Pattern.

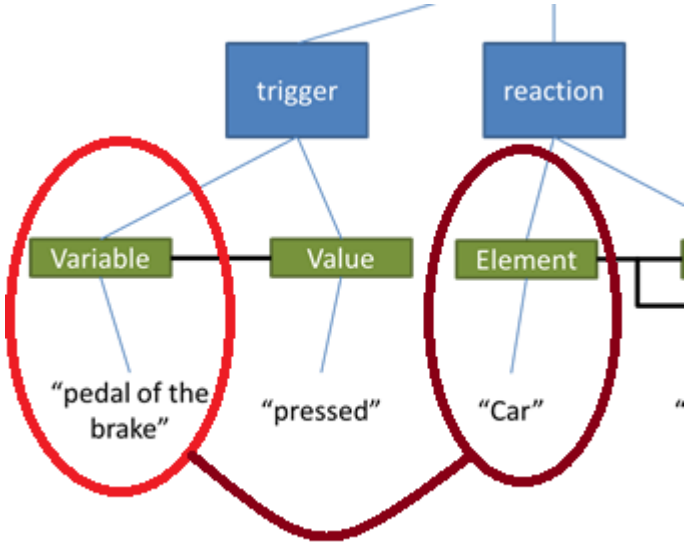


Figure 4-23. Formal transformation #1

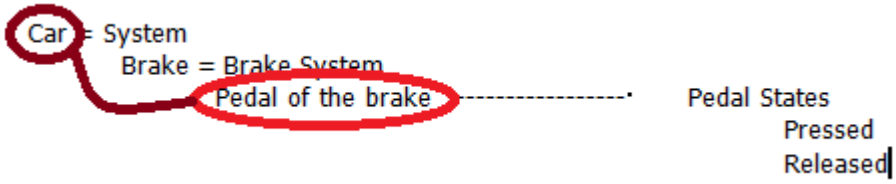
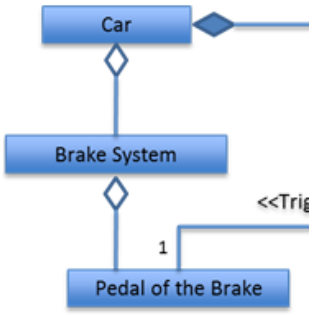


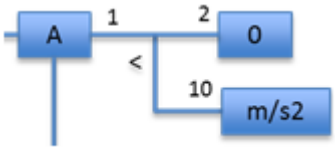
Figure 4-24. Restrictions for transformations

This restriction must be fulfilled in order to continue with the T2 transformation

If the restriction is fulfilled, the whole path between both terms in the ontology will be included in the graph representation of the requirement:



2- If a relationship type of one relationship is of semantic type <<reaction>> and the node with concept order 2 in the relationship is the term Decelerate occurs → substitute the node for

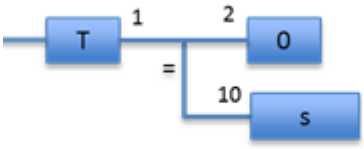


And the relationship type to a Composition of type “Property”

3- If a relationship type of one relationship is of semantic type <<reaction>> and the node with concept order 2 in the relationship is any child of a Physical Environment that has a Measurement Unit equivalence → substitute the term by the Measurement Unit equivalence.

4- If a relationship type of one relationship is of semantic type <<reaction>> and the node with concept order 2 in the relationship is any child of a Physical Environment that has NOT a Measurement Unit equivalence → Look for a relationship of Antinomy and check that exists (if it exists)

5- Substitute the Immediately term always by the graph



Connected to the Physical Environment Measurement Unit equivalence, forming

4.3.2 Story Example 2

The second example will show how we can use information extracted from SysML to improve the quality of requirements. As an example we will use two SysML diagrams from a Doors Management System, which was used as an example in the Cesar project.

The first diagram is an Activity diagram, which describes the activities performed after a passenger tries to unlock the door.

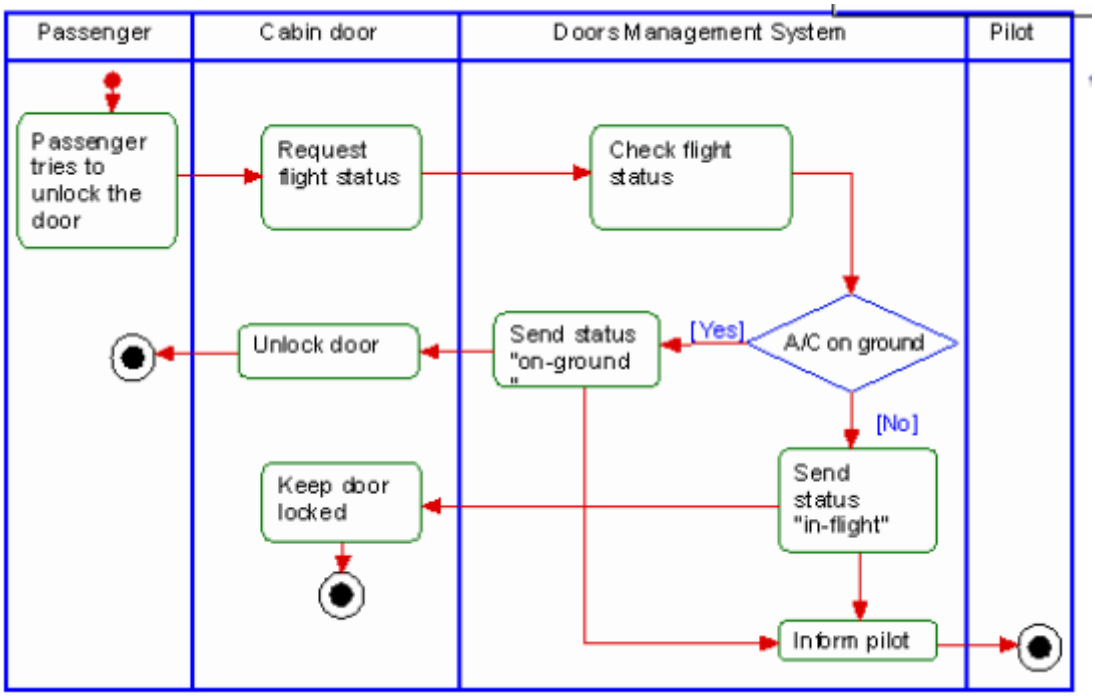


Figure 4-25: Example of SysML Activity diagram

The second example is the Use case diagram describing the Doors Management System.

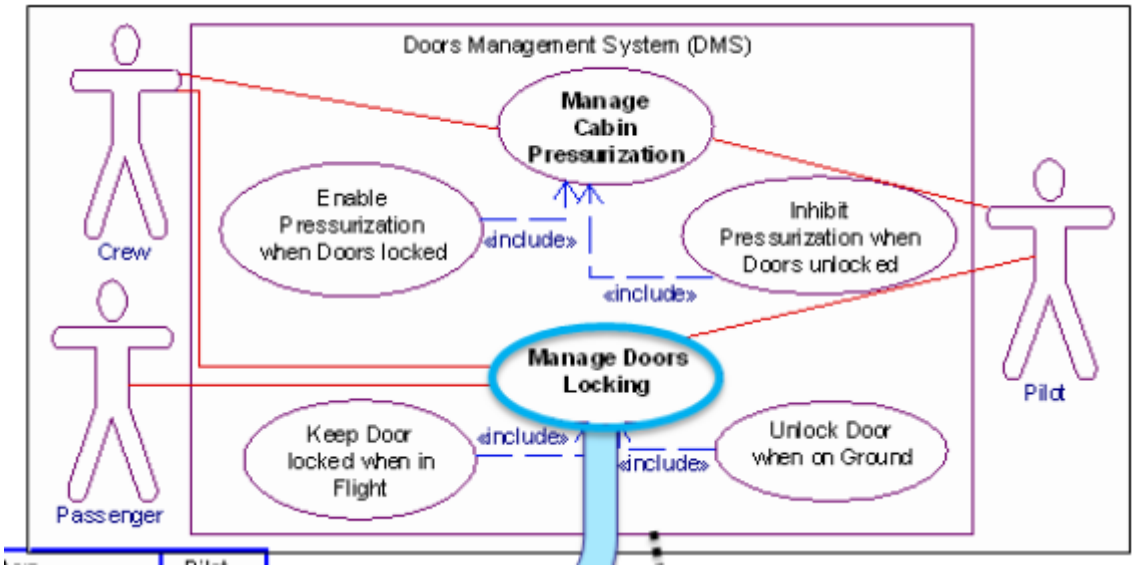


Figure 4-26: Example Use case diagram of Doors Management Systems

These diagrams can now be feed to the ontology, and the information can be used to check the requirements.

For example, if the ontology could include a list of User. We could now ask Completeness questions: Is there at least one requirement for every Stakeholder?

Another example would be to hold a relation “perform activity” between users and activities. This could help to find inconsistencies in requirements. For example the requirement

The cabin door should check the flight status

would be inconsistent with the diagram and should be marked as such.

4.4 Services to be provided through IOS

This section describes the services that RBE Work-package will contribute to the IOS platform. Mainly, these services will be around the two main contributions of this bricks to CRYSTAL: Quality management for requirements and ontology management and support.

4.4.1 Quality Analysis

This service deals with the configuration, analysis and reporting of the quality CCC criteria defined in different sections of this document.

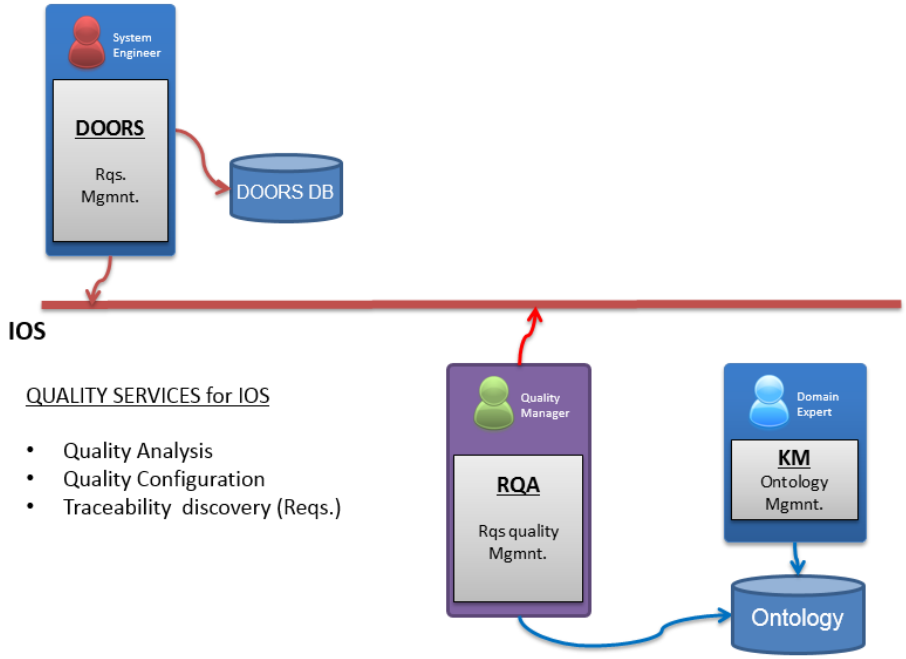


Figure 4-27. Quality IOS Service

4.4.2 Ontology services / Naming services

This service will allow other services to access to the current content of the ontology. One possible need for that is checking the names given to different elements created in other engineering domains not directly related to requirements engineering; for instance, checking for the proper names in a SysML model created in Rhapsody.

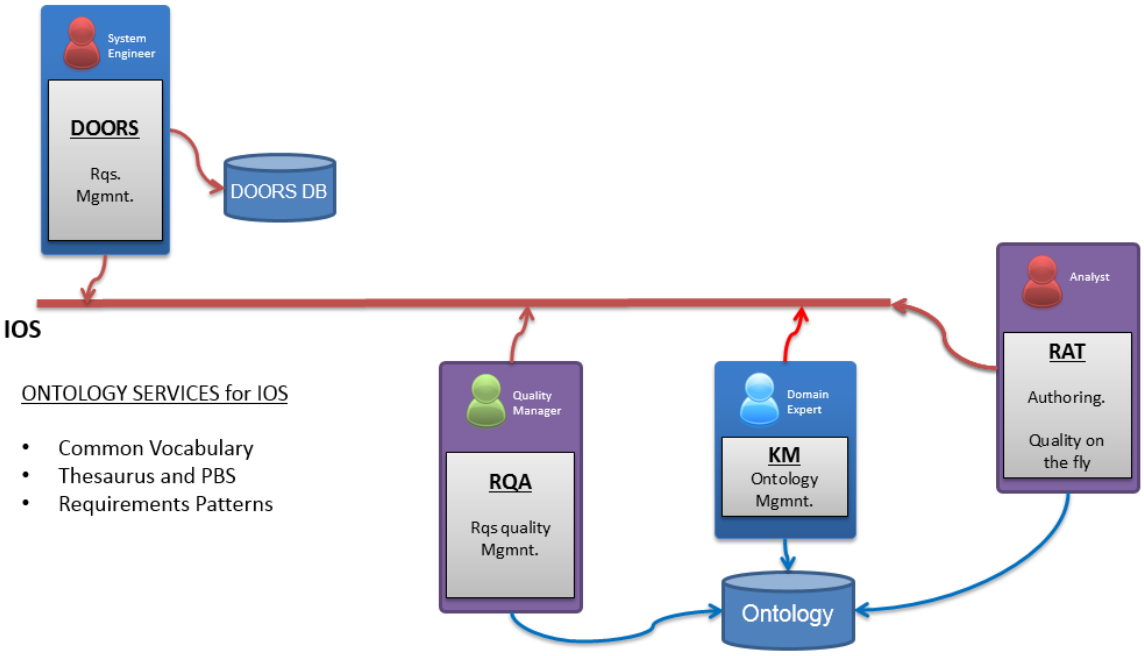


Figure 4-28. Ontology IOS services

4.4.3 Textual asset formalization

By means of this service, every textual asset (not only requirements, but also test cases, risk definition…) can be formalized by means of semantic graphs and meta-properties (see the semantic transformations described above, Figure 2-9). Once a textual asset has been formalized, any customized correctness metric can be applied (CORR3) and furthermore, it'll be ready to be retrieved by the semantic search engine.

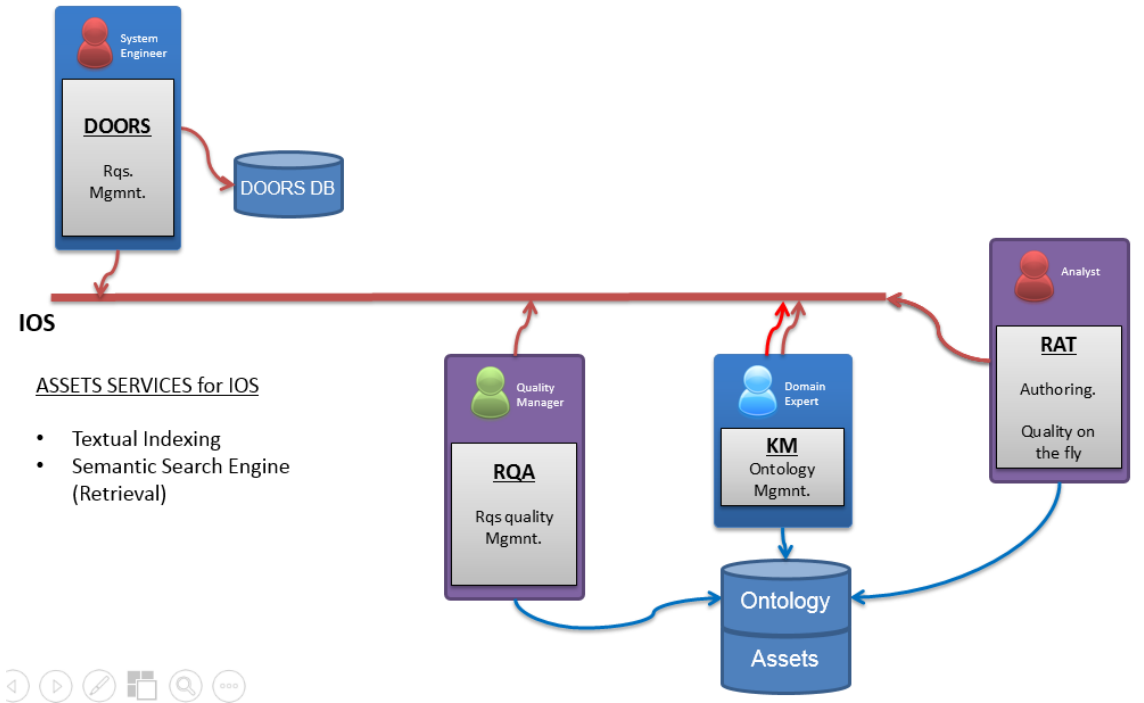


Figure 4-29. Textual asset formalization IOS service

4.4.4 Universal asset formalization

This service will extend the indexing and retrieval approach mentioned above to other non-textual assets such as SysML models, simulation models…

Through this service, any third-party will be able to write a code for properly transforming any kind of assets into the RSHP models described above.

Other possible input for this service is RDF-based content, where third-parties are able to generate a RDF file out of the content of their artefacts. This service will be able to import this RDF into the Asset Store of the Ontology.

Once an assets, textual or not textual, has been formalized into the System Asset Store (SAS), it will be available for the semantic search engine service.

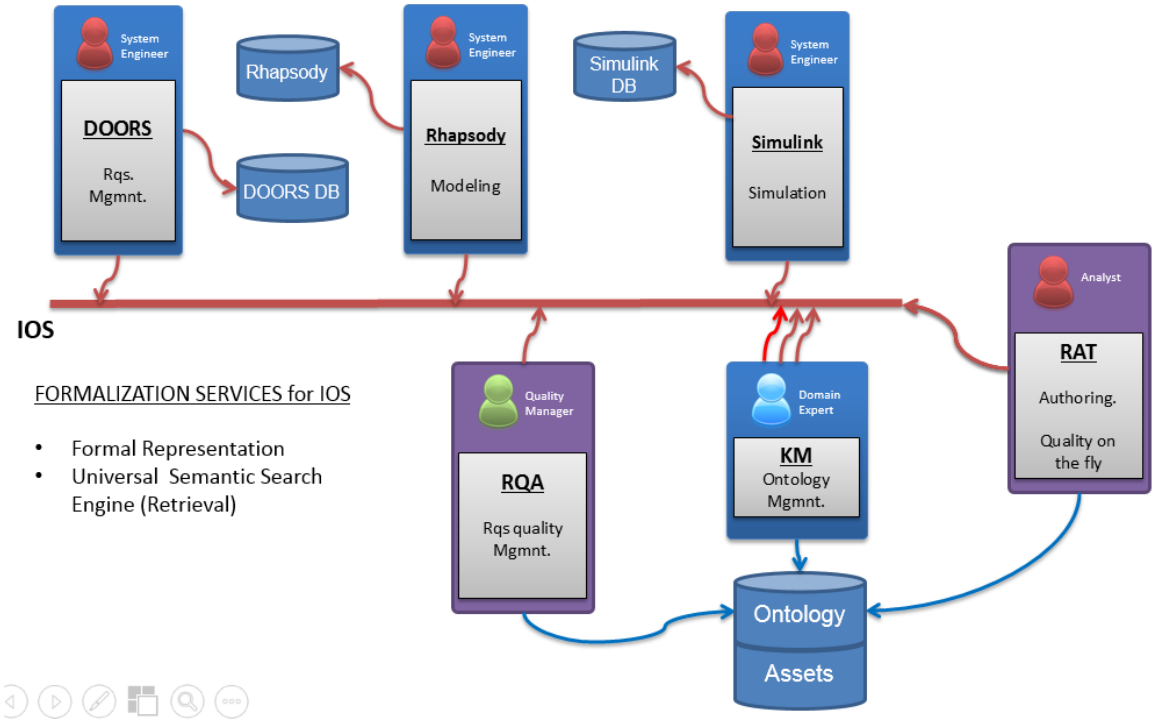


Figure 4-30. Universal formalization IOS service

4.4.5 Universal semantic retrieval

Together with the indexing processes, both for textual assets, but also the universal indexer, a universal semantic retrieval service will be provided, allowing to extend the scope of retrieval and reuse to both textual and non-textual kinds of assets (mainly models).

This service will provide several interfaces, including each of them a higher load of semantics. Examples of those interfaces could be:

- Keyword-based interface: the classical google-like approach, but using valuable information from the System Conceptual Model to enrich the query, providing better user experience
- Text-based interface: allowing a whole sentence as a query. In this case, the formal representation shown in previous sections will be performed for the query text. The results of the query will be based on the similarity of the semantic graph *query vs indexed assets*

- Similarity interface: a suitable artefact is provided as input. The service will generate the semantic graph for this kind of artefact and check for similar artefacts (by content, by type…)

5 Terms, Abbreviations and Definitions

5.1 Abbreviations

CCC	Completeness, correctness and consistency
CESAR	Cost-efficient methods and processes for safety relevant embedded systems
CO	Confidential, only for members of the consortium (including the JU).
CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
D	Demonstrator
KM	Knowledge Manager
LTL	Linear Temporal Logic
NL	Natural Language
NLP	Natural Language Processing
O	Other
P	Prototype
PBS	Product Breakdown Structure
PLM	Product Lifecycle Management
PP	Restricted to other program participants (including the JU).
PU	Public
R	Report
RAT	Requirements Authoring Tool
RE	Restricted to a group specified by the consortium (including the JU).
RMM	Requirements Meta-Model
RQA	Requirements Quality Analyzer
RQS	Requirements Quality Suite
RSL	Requirement Specification Language
SCM	System Conceptual Model
SKB	System Knowledge Base
TRC / REUSE	The Reuse Company
TRL	Technology Readiness Level
VIT	Virtual Integration Test
WP	Work Package

Table 5-1: Terms, Abbreviations and Definitions

5.2 Definitions

Boilerplate/Plate	A sequential list of restrictions stating the grammar (structure) of requirements and other types of textual artifacts. Such *boilerplates* are the core of for formalization steps needed to accomplish with most of the CCC criteria. More detail in section 2.1.3.

	A pattern can be seen as the instantiation (e.g. with actual data coming from a requirement) of a requirement. However, different approaches analysed during this project follow a different distinction between these two concepts. This clear role of *boilerplate* vs *patter* will be further clarified in future versions of this document
Ontology	Represented in different layers containing the specification of a conceptual model based on concepts, their meaning, different dimensions of classifications and the *boilerplates* needed to better represent (formalise) textual artifacts such us requirements. More detail in section 2.1.2

6 References

Please add citations in this section.

[Farfeleder,2011]	Farfeleder, Stefan, et al. "DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development." *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2011 IEEE 14th International Symposium on*. IEEE, 2011.
[CESAR_CCC,2011]	CESAR Deliverable Completeness/Consistency/Correctness D_SP2_R3.3_M3_Vol4 http://www.cesarproject.eu/fileadmin/user_upload/CESAR_D_SP2_R3.3_M3_Vol4_v1.000_PU.pdf
[CESAR_RSL,2011]	CESAR Deliverable Definition and exemplification of RSL and RMM D_SP2_R2.1_M3 http://www.cesarproject.eu/fileadmin/user_upload/CESAR_D_SP2_R2.3_M3_v1.000_PU.pdf
[SPEEDS, 2008]	SPEEDS Project, Speculative and Exploratory Design in Systems Engineering, http://www.speeds.eu.com/
[HRC, 2009]	Speculative and Exploratory Design in Systems Engineering (SPEEDS); SPEEDS L-1 Meta-Model; Deliverable D2.1.5; Revision 1.0.1; 2009-05-15; http://speeds.eu.com/downloads/SPEEDS_Meta-Model.pdf
[Gruber, 1993]	Gruber, T. R., “A Translation Approach to Portable Ontology Specifications”. Knowledge Acquisition, 5(2):199-220, 1993
[Guarino, 1995]	Guarino, N. “Formal Ontology, Conceptual Analysis and Knowledge Representation”. International Journal of Human-Computer Studies, 43(5-6):625–640, 1995
[INCOSE]	http://www.incose.org/ProductsPubs/products/guideforwritingrequirements.aspx (last visited 30.December.2013)
[Hull et al., 2011]	E. Hull, K. Jackson and J. Dick. (2011). *Requirements Engineering, 3rd. Edition.* Springer.
[Llorens et al, 2004]	Llorens, J. Morato, J. Genova, G. “RSHP: An information representation model based on relationships.” In: Ernesto Damiani, Lakhmi C. Jain, Mauro Madravio (Eds.), SoftComputing in Software Engineering (Studies in Fuzziness and Soft Computing Series, Vol. 159), Springer 2004, pp 221-253.
[RQA]	CRYSTAL Deliverable: CRYSTAL_D_607_021