

**SatTerm Project:
Vocabulary Control and Facet Analysis
help improve the Software
Requirements Elicitation Process**

Ricardo Eito-Brun
Universidad Carlos III de Madrid

ISKO UK CONFERENCE
London, July 8th -9th , 2013

Introduction

Terminology and linguistic resources in SW Engineering

- Terminology management and control is a critical area in the production of technical documents and software specifications
- Terminology control is needed to:
 - Help authors and reviewers use a common, defined set of terms.
 - Standardize the writing of technical documents among the writers in the team.
 - Ensure a common understanding with customers and partners of the terms used in different publications.
 - Avoid misinterpretations of potentially ambiguous terms.

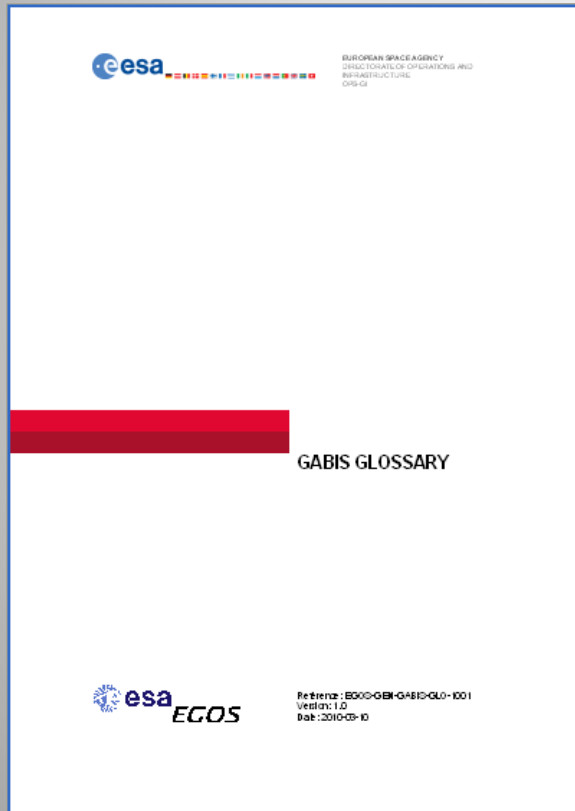
Introduction

Terminology and linguistic resources in SW Engineering

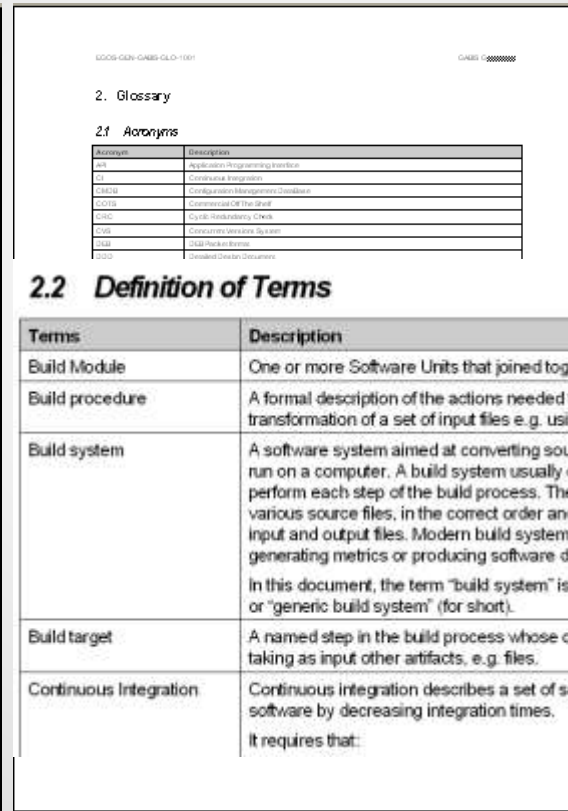
- Sample Use Case:
 - Engineers can access terminologies when writing technical documents to ensure the use of preferred terms in the right context.
 - Glossaries and acronyms are added at the beginning of technical documents to help readers understand their content.
 - Engineering and support teams access terminologies when writing or answering problem descriptions or ECR for searching and indexing.
 - The elaboration and publishing of glossaries is a common practice in major projects in the aerospace industry.

Introduction

Terminology and linguistic resources in SW Engineering



The cover of the GABIS Glossary document features the ESA logo at the top left and the text "EUROPEAN SPACE AGENCY" and "DIRECTIONAL OF OPERATIONS AND INFRASTRUCTURE" at the top right. A red horizontal bar is positioned above the title "GABIS GLOSSARY". At the bottom left, the ESA logo is accompanied by the text "EGOS". At the bottom right, the reference information "Reference: EGOS-GABIS-GLO-1001", "Version: 1.0", and "Date: 2010-03-10" is provided.



The content pages of the GABIS Glossary document show the title "EGOS-GABIS-GLO-1001" and "GABIS GLOSSARY" at the top. Section 2, "Glossary", includes a sub-section 2.1 "Acronyms" with a table of abbreviations and their descriptions. Section 2.2 "Definition of Terms" includes a table defining key terms like "Build Module", "Build procedure", "Build system", "Build target", and "Continuous Integration".

2.1 Acronyms

Acronym	Description
AP	Application Programming Interface
CI	Continuous Integration
CMDB	Configuration Management Database
COTS	Commercial Off the Shelf
CRD	Cycle Redundancy Check
CVS	Concurrent Versions System
DB	Database
DDO	Document Control Document

2.2 Definition of Terms

Terms	Description
Build Module	One or more Software Units that joined together implement a Software Subsystem.
Build procedure	A formal description of the actions needed for generating one or more output artifacts, usually by transformation of a set of input files e.g. using a compiler.
Build system	A software system aimed at converting source code files into standalone software artifact(s) that can be run on a computer. A build system usually coordinates and controls other programs and tools that perform each step of the build process. The basic function of the build system is to compile and link the various source files, in the correct order and taking into account the dependencies that exist between input and output files. Modern build systems include also functions for automatically analysing code, generating metrics or producing software documentation. In this document, the term "build system" is used to refer to the "Generic GSI Application Build System" or "generic build system" (for short).
Build target	A named step in the build process whose objective is the generation of one or more artifacts, usually taking as input other artifacts, e.g. files.
Continuous Integration	Continuous integration describes a set of software engineering practices that speed up the delivery of software by decreasing integration times. It requires that:

Introduction

Are current practices enough?

- Glossaries are a valid tool to support engineering tasks, but...
 - As the complexity of the working processes grows and
 - Knowledge Representation and Organization techniques improve,

Additional opportunities because

better knowledge-based supporting tools are available.

How can we take advantage of them?

Introduction

Are current practices enough?

- Three main areas of improvement in Requirements Management:
 - Generation of models / diagrams from requirements written in natural language (Model Driven Design/MDD).
 - Automatic Verification of requirements.
 - Reuse of requirements
- But...
 - We need to improve the way we represent / encode knowledge on requirements text.

Introduction

Are current practices enough... for MDA?

- SW development is a knowledge-intensive process.
- It requires the application of expert knowledge to solve the problem initially stated by the future users of SW.
- To formulate and develop the solution it is necessary to capture and encode knowledge about:
 - The problem and its context,
 - Specify, model and represent the features of the planned solution using different knowledge representation techniques (natural language, formal or semi-formal languages, diagramming techniques or the source code written in a specific programming language).
- SW development can be seen as a **transformation process** between different artifacts that represent the knowledge captured at the different steps of the process (MDA/MDD approach and technologies).

Introduction

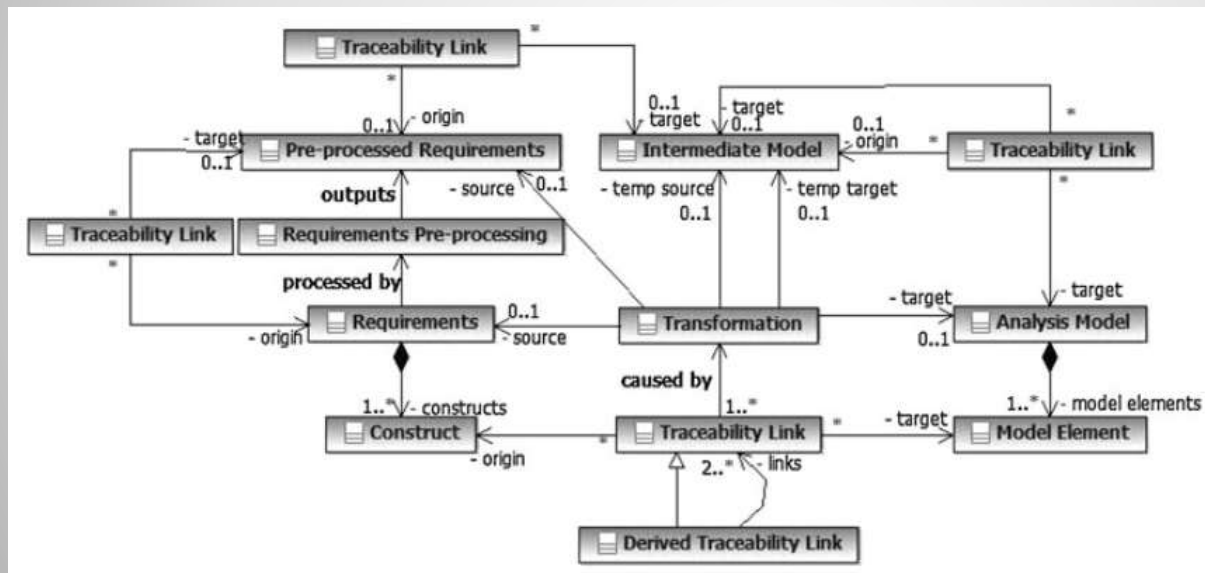
Are current practices enough?

1. Requirements
 - Written in natural language
 - Document-model
2. Design document (PIM)
 - Written using a modelling language (UML)
 - Diagrams that represent entities, relationships, properties.
3. Design document (PSM)
 - Written in a specific programming language (C, Java, etc.)
 - Implements the items identified in the design document.
4. Source code
 - Written using a programming language (C, C++, Java)
 - Derived from the PSM
5. SW Application
 - Files installed on the target computer.
 - Support the SW end-users activities and work processes

Introduction

Are current practices enough... for Model Driven Engineering?

- Is it possible to obtain an initial model (Platform-independent model) from a requirements specification written in natural language?

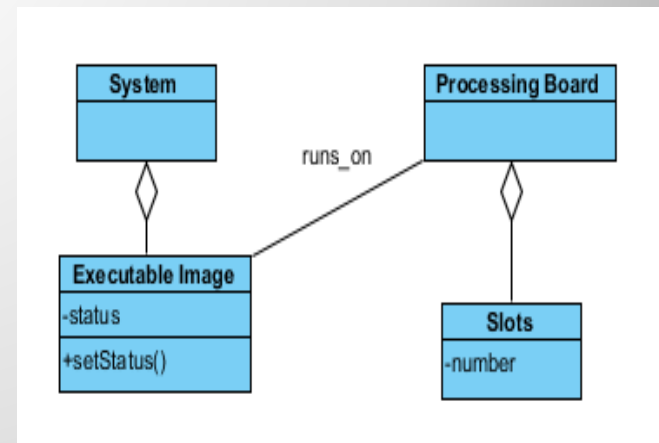


Yue, T., Briand, L.C. and Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. IN Proceedings of Requirements Engineering (2011), p. 75-99.

Introduction

Are current practices enough... for Model Driven Engineering?

- In order to automatically generate a design model from a textual specification, the “tagging” of the requirements text is needed to know whether the terms/words correspond to actors, processes, objects, measurements, numeric values, etc.
- *“The system executable image shall transition to FAILED if the board slot number on the processing board upon which it runs does not mach the 6.784 value”*
 - Entities: “system executable image”, “processing board”, “slot in processing board”
 - Agent: “system executable image”
 - Properties: “board slot::number”, “system executable image::status”
 - Actions: “transition to a specific status”
 - Condition: { board slot.number < 6.784 }
 - Contextual information: “system executable image runs on processing board”.



Introduction

Are current practices enough... for requirements verification?

- Requirement: a verifiable statement regarding some property that a software system has to possess.
- Requirements must fulfill some quality characteristics or attributes:
 - Completeness
 - Coherence
 - Lack of ambiguity
 - Verifiable.
 - No redundant, no duplicated requirements.
- The possibility of having a “knowledge base” with statements about the constraints and restrictions in the application domain (that is to say, the environment where the SW will be used), would make possible the verification of these attributes.

Introduction

Are current practices enough... for Requirements verification?

- Example:
 - If the knowledge base includes statements like this one:
 - *“Status limit is used to ensure that a digital parameter value is one in a list of values statically defined in the database...”*
 - It would be possible to identify potential errors like:
 - *“Parameter X is an analogue parameter with a status limit in between...”*

Introduction

Are current practices enough... For requirements reuse?

- “Knowledge reuse in software development processes” has replaced “software reuse”.
- Traditional SW reuse has focused on the reuse of source code, and on the creation of catalogues of software components (sometimes described by means of faceted classifications)
- Current trends on software reuse try to achieve the reuse of intermediate artifacts (software specifications, users requirements, design models, etc.) generated during the SW development process.
- Requirements can be “reused”:
 - To facilitate the retrieval and reuse of the specifications as an independent artifact.
 - To easily identify the design and code that implements these requirements.
- Tradicional “cataloguing” of requirements may be improved with more detailed descriptions of the requirements purpose and scope.

How requirements represent knowledge?

- In order to generate models, verify automatically requirements or reuse them, we need to improve the way we encode knowledge on requirements.
- The information in the requirements document is recorded on statements written in a specification language:
 - Free, non-restricted natural language,
 - Structured or restricted natural language,
 - Semi-formal methods and
 - Formal languages like VDM or Z.

How requirements represent knowledge?

- Restricted Natural language defines rules that govern:
 - the way engineers write the different types of statements (writing conventions to represent the conditional behavior of the system)
 - ensure that all the concepts used in the statements are properly defined before they are used,
 - do not include more than one requirements, hypothesis of domain feature in each sentence,
 - keep statements and sentences short,
 - use "shall" for statements that correspond to mandatory requirements and "should" for desirable requirements,
 - avoid acronyms and terms difficult to understand,
 - use examples to clarify abstracts declarations, g) include diagrams to represent complex relationships between elements,
 - avoid complex combinations of nested conditions that may lead to ambiguity, etc..
 - the structure of the requirements document.

How requirements represent knowledge? – Retrospective Analysis

- **Durán Toro** (1999) proposed an approach that combined “requirement patterns” (R-patterns) – and linguistic patterns (L-Patterns).
- L-patterns are frequently used types of sentences that can be combined to describe scenarios or interactions;
- L-patterns distinguish variable textual fragments that can be replaced with the appropriate terms when writing the specification.
- When writing requirements, engineers will use a template that describes interactions made up of different statements based on L-patterns; the engineer must make the necessary changes in these L-patterns to complete the description of the target interaction.

How requirements represent knowledge? – Retrospective Analysis

- Fantechi (2002) applied NLP techniques to detect defects due to the ambiguity of natural language in requirements documents
- Statements are analysed from the lexical, syntactical and semantic dimensions, as all these factors may have an effect on ambiguity.
- The analysis detects sentences that may be interpreted in more than one way and sentences with a complex structure.
- Different NLP tools were combined: QuARS, ARM y SyTwo.

How requirements represent knowledge? – Retrospective Analysis

- Tjong (2006) proposes to reduce the problem of language ambiguity and lack of precision by means of “language quality patterns” and “guiding rules”.
- The use of connectors like and, or, but, and/or and both usually result in ambiguous sentences.
- He proposed:
 - A set of patterns called GAND (Generic AND Pattern), GOR (Generic OR Pattern), IFFP (If and Only If Pattern), CACP (Compound AND Condition pattern), GP (Generic Pattern), GNP (Generic Negative Pattern), ECP (Event Condition Patterns) and TP (Time Patterns).
 - 15 writing rules : use positive sentences with a single verb [Rule 1], avoid passive verbs [Rule 2], avoid terms like “either”, “whether”, “otherwise” [Rule 4], “eventually”, “at least” [Rule 5], use “at most” and “at least” instead of “maximum” and “minimum” [Rule 6], avoid “both”, [Rule 7], “but” [Rule 8], “and/or” [Rule 10], “not only”, “but also” [Rule 11], etc. Rules 13, 14 and 15 refer to the need of fixing a glossary and a list of acronyms and abbreviations.
- To verify this approach, requirements documents from different domains were re-written applying these rules and patterns.

How requirements represent knowledge? – Retrospective Analysis

- Videira et al. (2006) presented a language for requirements specification based on the identification of frequently used linguistic patterns.
- The rules in the language were derived from an analysis of existing specifications.
- Most of the sentences follow the pattern “subject executes and action – expressed by a verb – that affects an object”.
- This analysis led to the development of a metamodel with:
 - Actors – active resources like external systems, end users, which execute actions on one or more entities.
 - Entities – static resources affected by the operations. They have properties that describe their status.
 - Operations – these are sequences of single actions that affect the entities and their properties.

How requirements represent knowledge? – Retrospective Analysis

- Boyd (2007) proposed to restrict both the vocabulary and the syntax to reduce ambiguity and complex sentences;
- Boyd presented an automatic approach and the concept of replaceability.
- Replaceability is defined as the possibility of replacing one term X with other term Y in a particular domain.
- It is calculated from the lexical similarity and polysemy.
- The information about terms replaceability is kept in two-dimensional matrices. For each term its Part of Speech is indicated as well as its meaning (this is an index that points to the exact meaning of the term in the document among all the possible meanings).
- One interesting aspect of this approach is the possibility of combining the identification and selection of terms by engineers with this technique.

How requirements represent knowledge? – Retrospective Analysis

- Ketabchi (2011) proposed the use of restricted natural language.
- Two steps:
 - Domain analysis and modeling: based on the analysis of work processes, activities and participants.
 - Definition of the problem space.
- Business rules were gathered using structured patterns, like, for example:
Whenever <condition>
if <state> then <agent> is <deontic-operator> to <action>.

How requirements represent knowledge? – Retrospective Analysis

- Majumdar (2011) proposes a formal syntax to write requirements using a restricted natural language called ADV-EARS.
- **To enable the automatic generation of use case diagrams from textual requirements it is necessary to apply some control on the requirements' text.**
- Requirements must be written following the restricted syntax, and then an automatic analysis is done to generate syntactic trees for each statement and to identify actors and use cases.

How requirements represent knowledge? – Retrospective Analysis

Req Type	Definition in EARS	Definition in ADV-EARS
UB	The <system name> shall <system response>	The <entity> shall <functionality> The <entity> shall <functionality> the <entity> for <functionality>
EV	WHEN <optional preconditions> <trigger> the <system name> shall <system response>	When <optional preconditions> the <entity> shall <functionality> When <optional preconditions> the <entity> shall perform <functionality> When <entity> <functionality> the <entity> shall <functionality>
UW	IF <optional preconditions> <trigger>, THEN the <system name> shall <system response>	IF < preconditions> THEN the <entity> shall <functionality> IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> to <functionality> IF < preconditions> THEN the <functionality> of <functionality> shall <functionality> to <functionality> and <functionality>
ST	WHILE <in a specific state> the <system name> shall <system response>	WHILE <in a specific state> the <entity> shall <functionality> WHILE <in a specific state> the <functionality> shall <functionality>
OP	WHERE <feature is included> the <system name> shall <system response>	WHERE <feature is included> the <entity> shall <functionality> WHERE < preconditions> the <functionality> shall <functionality> WHERE < preconditions> the <functionality> of <functionality> shall <functionality> to <functionality>
HY	Not defined	<While-in-a-specific-state> if necessary the <functionality> shall <functionality> <While-in-a-specificstate> if necessary the <entity> shall perform <functionality> <While-in-a-specific-state> if <preconditions> the <functionality> shall <functionality>

SatTerm Approach

Linguistic Patterns and vocabulary control

- Research has focused on the use of “linguistic patterns”, e.g. “syntactic patterns”, combined to model “typical interactions” and use cases.
- These linguistic patterns are useful to identify actions, agents, objects, conditions and roles, to make the transition from text to design models easier.
- The use of instruments like ontologies (or any other form of controlled vocabulary) may improve the quality of software specifications based on linguistic patterns.
- Ontologies – shared conceptualizations of knowledge in a specific domain -, provide a common understanding of the concepts and the relationships between them, and can be used to “fill in the blanks” in the linguistic patterns.
- Ontologies are better than other “vocabulary control tools”, as they allow engineers to model any kind of relationships between entities.

SatTerm Approach

Linguistic Patterns and vocabulary control

- Examples:
 - “Out of range is an event packet generated when a telemetry parameter raw value is outside of the defined calibration range”.
- This sample definition deals with:
 - Entities (Classes): Event Packets, Out of Range, Telemetry parameters, Calibration Ranges.
 - Properties: Telemetry parameter::raw value.
 - Specialization relationships: “Out of range” IS A “Event packet”.
 - Relationship: parameter::raw value IN calibration range
 - Actions: “packet generation”
 - Events/Conditions: Action happens when something happens (e.g. specific value is met)

SatTerm Approach

Linguistic Patterns and vocabulary control

- SatTerm is an academic research project that tries to combine the capabilities of ontologies with constrained natural languages to support engineers in the creation of specifications for Satellite Control software.
- SatTerm intends to combine EARS-based linguistic patterns with an ontology that contains domain specific knowledge and vocabulary.
- The collection of terms to build this ontology has been made from a set of existing software specifications.
- The resulting vocabulary includes around 400 terms (Monitoring, Control and Ranging).
- With this tools, requirements' text can be tagged to identify actors, actions, methods and tools, objects, properties, events and constraints.

SatTerm Approach

Linguistic Patterns and vocabulary control

- AGENTS:
 - Actor executing a process or requesting the execution of a task to the target system.
 - The inclusion of a specific entity within this group implies that the entity has the capability of doing something independently.
 - Components of the SW product tree can play this role.
- OBJECTS:
 - Items affected by or processed (inputs, outputs) of the tasks implemented by the target system.
 - This category is divided into two sub-categories:
 - Domain components: items that correspond to the reality the SW “represents”. Usually they are the source of data (data refers to their properties)
 - System components: hardware and software items that may be further subdivided into is-part-of relationships. For example, the ground system is composed of baseband equipment, up-converter, high power amplifier, etc. They receive, process or provide the operational data.
 - Operational data, includes the main data elements received, processed or generated by the system, as packages, telemetry, telemetry parameters, commands, command arguments, radiofrequency signals, alerts, messages, etc.

SatTerm Approach

Linguistic Patterns and vocabulary control

- OBJECTS:

- These classes are further subdivided by specialization.

Example: telemetry parameters may be subdivided into acquired or derived (the last ones are calculated taking acquired parameters as inputs) synchronous or asynchronous, etc. Different criteria are applied to classify the objects of the same class (e.g.: origin, need of post-processing, etc.)

- The ontology includes properties linked to classes. Properties make possible a better description of entities.
- **Properties** also allow the detailed specification of the rules that govern the behavior of the target system and the events to which the target system must respond.

Example, telemetry parameters share properties like the raw value, last recorded value, value obtained after interpolation, out of limit state or limit (values or range of values that, in case of being exceeded, should raise an event to generate an alert)

SatTerm Approach

Linguistic Patterns and vocabulary control

- PROCESSES/TASKS:
 - Activity or set of activities that generates an output from an input, making some kind of transformation or processing.
 - This category includes basic task: release, encode, encrypt, execute, verify, multiplex, archive, uplink, downlink, configure, authenticate, count, calibrate, retransmit, convert, calibrate, receive, etc.
 - Tasks are classified as datalogical (copy, store, transmit), infological (concerned about content) and Ontologicals (new, original things are brought about)
 - Tasks actuate on objects, for example, calibrate acts on telemetry parameters or multiplex acts on commands.
 - Basic tasks are combined into COMPLEX TASKS, usually following an order (sequence, process in parallel).
Example: telemetry chain process is made up of the following sequence of tasks (all of them executed on telemetry data): receive, packetize, archive, distribute, de-commutate.
 - A particular task is **verification**. Example, telemetry correctness may be verified **applying different techniques** (flow id checking, frame error control check, frame synchronization check or frame locking, synchronization work check or spacecraft Id check).
 - Verification activities usually act on other activities.

SatTerm Approach

Linguistic Patterns and vocabulary control

- PROCESSES/TASKS:
 - Actions and processes are defined by means of different “object properties”:
 - agent that executes them,
 - Entities used as input,
 - Entity, process or action on which the action is executed (inputs),
 - Resulting entities (outputs),
 - Agent or entity that receives the result of the action,
 - Constraints (rules and instruments) to be used when executing the action,
 - Pre- and Post-conditions (usually related to values of specific entity properties).
- CONSTRAINTS:
 - It includes the rules and guidelines to follow when executing a process, process step or task:
 - standards for data formats and communication protocols (e.g. PCM, CORTEX, COP-1, etc.),
 - data transformation methods (Gray code, reverse bits, etc.),
 - calibration and verification methods (analogue calibration, digital or textual calibration, polynomial calibration, linear discrete calibration, etc.)

SatTerm Approach

Linguistic Patterns and vocabulary control

- EVENTS:
 - Situation that are notified to (or identified by) the target system, that require particular attention and some kind of response.
 - Events may be originated in the system environment (e.g. Infrared Earth Sensors Inhibit period or Eclipse period) or in response of an anomalous state of one of the objects monitored by the system (e.g. one telemetry parameter value is out of range).
 - Events represent one of the most interesting aspects for modelling, as they establish a relation between the condition that triggers the event, and the action to be executed in response to the event.
- TIME:
 - Most of the aspects and data managed by these systems are time-dependent (e.g. commands may be planned to be executed at a specific time, time synchronization between the system components is needed, etc.).
 - The time or period when an action – or the response to an event - has to be done is relevant and is usually included in the specification of requirements.
- MEASURES

SatTerm Approach

Linguistic Patterns and vocabulary control

- CONCEPT TYPES AND ROLES:
 - In the classification of the concepts, the main categories listed before have been applied.
 - Entities within the category OBJECTS may play different roles:
 - They can be the object of an action (the action is done on then),
 - **Telemetry Packet** is stored.
 - Polynomial calibration algorithm calculates the **OOL** with formula...:
 - They provide support the execution of an action (instrumental):
 - *Telemetry Packet is stored in **satellite onboard memory**.*
 - *The out of limit state of the parameter is calculated using **polynomial calibration algorithm**.*
 - They can be the target of the action:
 - *Telemetry Packet is stored in onboard memory to be processed for **uplink receptor**.*
 - Our analysis gives the choice of using the same concept (OBJECTS category) with different roles. The role is determined by the place of the concept in the linguistic pattern.

Conclusions

- The combination of controlled vocabularies with restricted syntax (linguistic patterns) to write software requirements is a promising area in product-line based projects.
- With an ontology that models the behavior, constraints, data and actions of a system (product or product-line) engineers can fill or add constraints to predefined sentence patterns (filling the blanks with terms taken from the ontology).
- Value of Actions:
 - It is not the same: “what this requirement is about” as...
 - “What is requesting this requirement?”
 - Analysis of requirements content must focus on actions|verbs.
 - Analysis must consider the context of the action: when is executed, how often, input data, output data, input format, output format, constraints, tools, agents,...
- These approaches ensure consistency in the specifications and give engineers an overview of the main concepts supported by the system.
- Modeling requirements in a structured way allows the automated verification of specifications and improve our capability to generate design models from text.

Next Steps

- Further verification of the ontology by third parties.
- Cover additional domain sub-areas (satellite navigation)
- Development of an XML-based editor to assist engineers when writing / tagging requirements.
- Improvement opportunities are the need of user-friendly interfaces to browse the ontologies and the need of reviewing the number of linguistic patterns.
- Guidance to users to select the linguistic patterns when writing requirements is also necessary.

- Collaboration is welcome!