SatTerm experience: vocabulary control and facet analysis help improve the software requirements elicitation process

Ricardo Eito-Brun

Universidad Carlos III de Madrid

Abstract

One of the most difficult steps in the software development process is moving from requirements written in natural, uncontrolled language, to the formalisms required by the design modelling languages. To solve this issue, practitioners should pay attention to the possibility of applying vocabulary control and knowledge representation techniques to produce better specifications. The use of controlled vocabularies and the modelling of the conceptual relationships between concepts in a specific domain are expected to improve the quality of the specifications. Vocabulary control and semantic modelling are promising tools to avoid the most frequent problems in the requirements specification process: lack of consistency and ambiguity.

This paper provides a detailed description of the development process of an ontology used for requirements modelling in the area of satellite control systems. The process applied is based on well-established practices and guidelines applied for the construction of controlled vocabularies and faceted classifications schemas. Engineers can use the ontology when writing system specifications using predefined templates. The use of this ontology ensures the consistency of the specifications written by different engineers improves the communication with other parties involved in the system construction activities and sets the foundations for a semi-automated generation of models for subsequent design activities.

1. Introduction

This paper describes an initiative to improve the elicitation of requirements in software-intensive systems. The approach is based on the use of vocabulary control techniques to improve the consistency of the specifications of software systems. Vocabulary control techniques help staff working on system engineering share a common set of terms and concepts in the different phases of the system development process. The objective of this work is to analyse how traditional vocabulary control may help improve the requirements engineering activities: requirements capture, analysis and verification.

Berztiss (2002, p. 121) defines requirements engineering as: "*Requirements engineering can be characterized as an iterative process of discovery and analysis, designed to produce an agreed-upon set of clear, complete, and consistent system requirements*". Lamsweerde, (2009, p. 6) offers a different definition as "*a coordinated set of activities for exploring, evaluating, documenting, consolidating, revising and adapting the objectives, capabilities, qualities, constraints and assumptions that the system-to-be should meet based on problems raised by the system-as-is and opportunities provided by new technologies*". ISO 9000:2000 and the IEEE glossary share this definition for requirements as "*Need or expectation that is stated, generally implied or obligatory*",

and Berztiss (2002, p. 121) defines a requirement as "*a verifiable statement regarding some property that a software system is to possess*".

As a discipline, requirements engineering is made up of the following activities (Lamsweerde, 2009, p. 30:

• Domain understanding: it consists on the study of the current system within its organizational and technical context, to learn the environment where the problem and its causes have been identified.

• Requirements elicitation: it focuses on the discovery of the requirements and hypothesis of the system to be developed. This activity explores the problem space with the key participants to acquire the necessary information about their objectives, constraints, scenarios and the desired interactions between the new software and its environment. Requirements elicitation applies techniques like interviews, prototyping, and analysis of existing documents and the direct observation of the work environment.

• Assessment and agreement: during the process, it is necessary to deal with the conflicts identified during the elicitation of the requirements.

• Specification and documentation: the agreed features of the target system needs to be detailed, organized and documented in a requirements document where the objectives, concepts, domain properties, responsibilities, requirements and hypothesis are explained.

• Requirements consolidation: its objective is to ensure the quality of the final specification. The specification will be analysed and validated by the key participants. The specifications also need to be cross-verified to identify inconsistencies before sending them to the programmers.

2. Knowledge Organization and Reuse in Software Development Processes

The term "requirements engineering" has replaced "requirements management". Similarly, there is a clear trend to use the term "knowledge reuse in software development processes" instead of "software reuse". Software development can be seen as a process that takes as an input the needs of the users. From these initial specifications, different intermediate artefacts are created applying different abstraction levels, to finally reach the software components and source code files that make up the definitive solution. This is a process where new artefacts are created based on existing ones, and different transformations are made between artefacts until the source code is ready. To do these transformations, engineers need to complete different intellectual activities and systematically apply a set of rules starting from artefacts with a high level of abstraction (user requirements) to the computer-dependent source code with the lowest level of abstraction. Software development methodologies propose the creation of different artefacts and the use of modelling techniques to present the results of these transformations and intellectual activities and to ensure the traceability among them.

Software development process requires the application of expert knowledge to solve the problem initially stated by the software target users. To formulate and develop the software-based

solution it is necessary to capture and encode the knowledge about the problem and its context, and to specify, model and represent the features of the potential solutions using different knowledge representation techniques (natural language, formal or semi-formal languages, diagramming techniques or the source code written in a specific programming language). Current trends on software reuse try to achieve the reuse of the intermediate artefacts (software specifications, users requirements, design models, etc.) to make the development of new software easier.

3. Requirements Modelling Techniques

The inputs of the requirements modelling process are the initial agreements on the new system objective and features, the hypothesis about the target environment and the concepts that characterize the domain where the software is going to be deployed. The output of the process is a requirements document that contains all this information structured in a specific way. The information in the requirements document is recorded on statements written in a specification language that makes possible the communication between the key participants and end-users and the engineering team. In an ideal situation, the specification language should support the automatic verification of the correctness, completeness and coherence of the set of statements that make up the specification. The specification languages in use include: free, non-restricted natural language, structured or restricted natural language, semi-formal methods and formal languages like VDM or Z.

The main advantage of natural language is its major expressiveness due to the lack of restrictions. It does not require any additional knowledge and does not establish any barrier between the participants in the process. Natural language drawbacks lie on its potential ambiguity, and on the complexity of making an automatic processing and analysis of statements, what makes the automatic verification of the specifications extremely difficult.

Restricted natural language goes a step further and defines rules that govern: a) the way engineers write the different types of statements and b) the structure that the requirements document must conform to. The first set includes writing conventions and decision tables to represent the conditional behaviour of the system. Some typical rules are summarized by Lamsweerde (2009, p. 121): a) ensure that all the concepts used in the statements are properly defined before they are used, b) do not include more than one requirements, hypothesis of domain feature in each sentence, c) keep statements and sentences short, d) use "shall" for statements that correspond to mandatory requirements and "should" for desirable requirements, e) avoid acronyms and terms difficult to understand, f) use examples to clarify abstracts declarations, g) include diagrams to represent complex relationships between elements, h) avoid complex combinations of nested conditions that may lead to ambiguity, etc. These simple rules may be combined with more complex syntactic and semantic restrictions in the elaboration of statements, like the ones described in Durán Toro (1999); Majumdar (2011) Tjong (2006) Renault (2009) Boyd (2007) Ketabchi (2011) or Fantechi (2002). Different approaches have proposed the use of restricted natural language as a method to facilitate the reuse and automatic verification of requirements.

A pioneer work is the one proposed by Durán Toro (1999) who proposed and approach that combined "requirement patterns" (R-patterns) – and linguistic patterns (L-Patterns). The concept of

pattern used by the authors differ from the concept of pattern proposed by Lam and other authors, who says a pattern is a set of related requirements that together implements a specific function of the system.

R-patterns are templates made up of a set of fields or data elements that guide engineers in the requirements elicitation process with the system's end users. L-patterns are frequently used 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. The textual fragments that can be replaced are called "customizable aspects of the linguistic pattern", and are written within the reserved characters < and >; The use of the characters { and } is also proposed to indicate the need of choosing one option among a set of allowed values. When writing requirements, the engineer will use a template that describe interactions made up of different requirements based on L-patterns; the engineer must make the necessary changes in these L-patterns to complete the description of the target interaction. Regarding the R-patterns, the proposed templates are based in turn in those described by Alistair Cockburn to document use cases. Having pre-defined templates to document recurrent use cases and requirements gives the choice of reusing the work done in previous projects and to streamline the specification of the user requirements.

Another study within this group – focused on the use of restricted natural language -, is Fantechi (2002), where natural language processing techniques are applied to detect defects due to natural language ambiguity in requirements documents base don use cases . Use case diagrams offer a summary view of the system capabilities, but they do not allow specifying the behaviour of the system, and natural language must be used to specify scenarios and extensions. Fantechi does not describe techniques to write the specifications, but to assess their quality from a linguistic perspective and alert engineers of potential problems and errors. A characterization of the quality of the textual requirements is done by collecting a set of metrics that are indicators of the expressiveness, consistency and completion of the specification. Use cases are made up of sentences that are analysed from the lexical, syntactical and semantic dimensions, as all these factors may have an effect on ambiguity (for example, one sentence may not be ambiguous from the syntactical point of view, as just one single tree can be derived from it, but it may be ambiguous semantically if it contains words with more than one meaning). The lexical analysis of the sentences is done to detect sentences that may be interpreted in more than one way, and the syntactic assessment is done to identify sentences with a complex structure. These analyses are done with different PLN tools: QuARS, ARM y SyTwo.

• QUARS (Quality Analyzer for Requirements Specifications) is based on a set of indicators that consist of terms and linguistic constructs that are representative of potential defects. QuARS analyzed the sentences and identify the presence of words that make the document ambiguous or complex, e.g.:

The C code shall be clearly commented

The system shall be as far as possible composed…

The system shall be such that… possibly without….

• ARM (Automated Requirements Measurement) is a tool developed by NASA that evaluates the specifications by searching for terms and fragments that may potentially led to errors.

• SyTwo is a web-based tool that analyses text to verify whether the sentence fulfils the rules of simplified English. It is an evolution of QUARS that calculates Coleman-Liau index and identifies syntactically ambiguous sentences, like for example: "The system shall not remove faults and restore service".

Another approach in this line of work was described in Tjong (2006). The author proposes to reduce the problem of the natural language ambiguity and lack of precision by means of language quality patterns and guiding rules. Tjong reported that the use of connectors like and, or, but, and/or and both usually result in ambiguous sentences. To avoid these situations, he proposed a set of patterns called GAND (Generic AND Pattern), GOR (Generic AND 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). The last one represents operations caused by events and conditions and requirements related to time. The guiding-rules are combined with the linguistic patterns to reduce ambiguity. A set of fifteen rules are described, among them: 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 be used in the specification of requirements. To verify this approach, different requirements documents from different domains were analyzed and re-written applying these rules and patterns, and the development of a tool called SREE (Systemized Requirements Engineering Environment) was started to help the analysts write their specifications. The requirements written using these rules could be analysed and tagged automatically to generate diagrams and analysis models.

Videira et al. (2006) presented a language for requirements specification called ProjectIT-RSL. This language was based on the identification of frequently used linguistic patterns. To language is supported by the PIT-Studio/RSL tool with functions to autocomplete, annotate, display potential errors, etc. With this tool, the engineer can write requirements documents and get alerts of potential errors and incorrect or ambiguous sentences. The rules in the RSL specification 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". Additional types of sentences may be built by adding conditions or specifying the attributes that define an entity. This analysis led to the development of a metamodel that included these elements: a) Actors – active resources like external systems, end users, which execute actions on one or more entities. b) Entities – static resources affected by the operations. They have properties that describe their status. c) Operations – these are sequences of single actions that affect the entities and their properties. Videira remarked the need of using a limited set of terms in the specification, and the control of the vocabulary

evolution by adding synonyms in a controlled way. The sentences are finally processed to generate RDF/OWL output to further develop semantic reasoning on the specifications.

The use of restricted natural language was also the subject of the study lead by Boyd (2007) who based his research on the Constrained Natural Languages (CNL) used in the elaboration of technical documents. These languages restrict both the vocabulary and the syntax with the purpose of reducing the ambiguity and keep under stability and expressiveness. Syntax constraints avoid complex sentences; the constraints in the vocabulary are useful to remove unnecessary variations and use only those terms with less ambiguity. Boyd remarked that the selection of the terms in existing CNL was based on the preferences of lexicographers, and not in a detailed analysis of the factors that may have an impact on the quality of the resulting documents. CNL are static languages that impede the addition of additional terms to those identified in the preliminary analysis; the terms in the CNL vocabulary are obtained from the analysis of text bases and corpus with the help of subject experts. Boyd presented an automatic approach to restrict the vocabulary of a CNL based on the semantic relationships between terms. To ensure the selection of the best terms (that is to say, those with less ambiguity) the author introduced the concept of replaceability. Replaceability is based on the lexical similarity that measures to which extent the meaning of two terms is similar with the purpose of finding redundant terms. Boyd proposed to use the replaceability instead of the term similarity, and limited its experiment to verbs. 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. A replaceability equal or greater than 1 means that a term can be replace with the other. One term will be replaced by another one only if it is used more often in the specification or if the polysemy (number of different meanings) is minor. 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, to ensure a better selection of terms and reduce, as much as possible, their potential ambiguity.

Renault et al., (2009a, 2009b) proposed another similar approach: PABRE, Pattern-Based Requirements Elicitation Method. This method was designed to write requirements based on the selection and integration of existing components or COTS (Commercial off the Shelf). PABRE was based on the experience of the Public Research Centre Henri Tudor (CPRHT), Luxembourg, to capitalize the knowledge acquired in past projects and to help transfer experience between projects. Until PABRE was designed, requirements reuse was done by duplicating requirements from existing projects, but this practice was not effective, and requirements were not normalized and they were domain dependent. Analysts should also know about the existing requirements in order to reuse them.

In PABRE, a catalogue of requirements patterns was made available, and new projects could derive their own requirements from this catalogue. The requirements that appear several times, with some variants, are considered as solutions to particular problems in a given context. Each pattern will have a set of descriptive metadata (name, description, author, objectives, projects where it was

used, keywords) and a form. The form has a fixed part (this is a brief sentence written in natural language that states what the system must do, not how to do that), and extensions that provide additional details, constraints and complementary information to the fixed part. The extensions are also text fragments that may have arguments or parameters, and it was possible to specify values for these arguments and parameters. PABRE requirements patters also had dependencies and were classified based on an schema created from the CPRHT experience. Renault mentioned that the size of the catalogue was 48 patters identified by the analysis of existing specifications. The approach was tested in a project to renew digital library software and in a CRM SaaS.

Ketabchi (2011) also proposed the use of restricted natural language. His concept of pattern is based on the definition proposed by Ambler (2000): "*a solution to a common problem, taking relevant forces into account and enabling the reuse of proven techniques and strategies*". This proposal is based on semiotics, discipline dedicated to the study of the signs and how they act on the society. One sign is something with a meaning, like a word, sound, object or an image. The sign as an object as a reference and there is an interpreter who interprets the sign following some guidelines or rules. Ketabchi indicates that – as part of requirements engineering – the use of the signs and the nature of the information that the users create, analyse, store and apply to communicate with other users should be agreed. Ketabchi's proposal is not the first one that relates semiotics and requirements management: the author mentions the MITAIS domain analysis technique that establishes a difference between the "decision space" and the "information space" as a relevant predecessor. In a similar way, when specifying requirements it is necessary a) to develop the decision space to express the problem to solve, and b) to set up the information space where the information needed by the specified problem is represented. Within this theoretical framework, Ketabchi establish an approach based on the use of problem patters linked to requirements patterns: each problem pattern identifies the requirements needed for its resolution after applying these steps:

a) Domain analysis and modelling: based on the analysis of work processes, activities and participants. The output of this activity is a set of problem-patterns that relates actors and activities with business rules and process guidelines.

b) Definition of the problem space and configuration of the requirements space. The problems are further detailed in a structured form and mapped to requirements.

These ideas were applied in the implementation of the library management software for the University of Reading Library, supported by UML use cases and sequence diagrams. Business rules were gathered using structured patterns, like, for example:

Whenever <condition>

if <state> then <agent> is <deontic-operator> to <action>.

Due to the importance of these rules, Ketabchi describes his approach as "norm-based".

Majumdar (2011) proposes a formal syntax to write requirements using a restricted natural language called ADV-EARS. The requirements written according to ADV-EARS rules could be analysed automatically to identify actors, use cases and to generate use case diagrams. To enable the

automatic generation of use case diagrams from textual requirements it is necessary to apply some control on the requirements' text. The author uses the term "syntactic structures for requirements" to refer to these constraints. His proposal is aligned to initiatives like the use of Simplified Technical English, ACE (Attempt to Controlled English), ECA (Event-Condition-Action) o EARS (Easy Approach to Requirements Syntax). In fact, ADV-EARS is an extension of EARS in which additional syntactic structures for different types of requirements are added as well as a context free grammar (CFC) for diagram generation. In ADV-EARS, 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. When more than one use case is identified in a single statement, the resulting use cases are related by means of an include relationship.

The predecessor of ADV-EARS, EARS, started from a classification of requirements into different groups: ubiquitous or nominal, non-desired behaviour, event-driven and status-driven. ADV-EARS added a new type called hybrid that combines the event-driven and conditional types to support the modelling of requirements that correspond to behaviours that start in response to an event plus a pre-condition. The syntax or set of sentence-types proposed by EARS was also extended to "accommodate additional models of sentences". The table below shows, for each type of requirement, the syntax proposed in EARS and ADV-EARS:

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>

Req Type	Definition in EARS	Definition in ADV-EARS
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>

Table 1: Sample patters in EARS and ADV-EARS (Majumdar, 2011, p. 60)

4. Semantic Support to Requirements Engineering

The analysis of existing bibliography gives the opportunity to identify complementary lines of work:

• Design of editing tools that give Access to controlled vocabularies when writing the specifications, and that verifies the linguistic quality of the specification applying rules like those described by Fancheti or Boyd.

• Establish methods to identify potential defects on requirements not only linked to their syntax, but with the conceptual and semantic information they contain. The semantic coherence within a set of requirements requires expert knowledge, and knowledge representation structures (e.g. ontologies) may be applied not only to capture what we know, but also to identify what it is not known and should be known by engineers working with specifications.

It is evident that the use of semantic instruments like ontologies may help improve the quality of software specifications. Ontologies – understood as a shared conceptualization of knowledge in a specific domain -, are key components in knowledge representation and sharing policies, and they provide a common understanding of the concepts and the relationships between the concepts used by end-users and engineering teams.

SatTerm is an academic research project that tries to combine the capabilities of ontologies with restricted natural languages to support engineers in the creation of requirements specifications for a specific domain: Satellite Control and Navigation software. SatTerm combines a predefined set of linguistic patterns (e.g. types of sentences) and an ontology that encodes domain specific knowledge. The generation of analysis models from textual descriptions elaborated with SatTerm would be possible, as the requirements' text clearly distinguishes the different actors, actions, operations, objects, properties and constraints. The transition to an analysis model is made easier, as the ontology already identifies what it is an entity of type actor, what is an entity of type object, what

are the objects' properties and which are the instruments and events the target system must deal with.

The collection of terms to build this ontology has been made from a set of existing software specifications. The relationships between terms, and the identification of classes and properties and they organization in different groups based on facets have been done manually by subject experts). The resulting vocabulary includes a total of 437 terms. Concept organization is based on the following schema or set of facets:

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. Attending to this criterion, this category may include entities that receive the results of the execution of a process by the target system (as long as they could make something with them).

OBJECTS: This category includes the items affected or processed (used as inputs) by the processes and tasks implemented by the target system. They can also be outputs of processes and tasks. This category is divided into two main sub-categories: a) system components – at different levels of granularity -, and b) operational data. The first group includes 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. The second group, 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. These classes are further subdivided by specialization: *telemetry parameters* for example 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.: provenance, need of post-processing, etc.)

The ontology includes properties linked to classes. Properties make possible a better description of entities and objects. They also allow the detailed specification of the rules that govern the behaviour of the target system and the events to which the target system must respond. For 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)

PROCESSES: activity or set of activities that generates an output from an input, making some kind of transformation. This category is further decomposed into BASIC TASKS, like for example *release*, *encode*, *encrypt*, *execute*, *verify, multiplex, archive, uplink, downlink, configure, authenticate, count, calibrate, retransmit, convert, calibrate, receive,* etc. 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 a sequencing constraint. For example, the *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 class of tasks and process are those related to verification activities. Verification is a key component in software implementations, and different types of data items, events and tasks may be the object of verification or quality checks. For 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). For commanding there are other verification activities: pre-transmission, uplink, execution, etc. One of the most interesting points of verification activities is that they usually act on other activities.

Actions and processes are defined by means of different object properties: the agent that executes them, the object, process or action on which the action is executed (inputs), the resulting object (outputs), the agent or object that will receive the result of the action, the constraints (rules and instruments) to be used when executing the action, pre- and post-conditions.

CONSTRAINTS: This category 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 are included within this group (for example, *analogue calibration*, *digital or textual calibration*, *polynomial calibration*, *linear discrete calibration*, etc.) Items in this category are usually organized through specialization relationships.

EVENTS: Events are situations that are notified (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. The trigger condition in turn usually implies a complex relationship between different components.

TIME: it plays a relevant role in this ontology, as most of the aspects and data managed by this type of systems are time-related (e.g. commands may be planned to be executed at a specific time, and time synchronization between the system components is necessary). The time or period when an action – or the response to an event - has to be done is relevant and it is usually included in the specification of individual requirements.

OWL (Ontology Web Language) supported by the Protégé software tool has shown effective to model the characteristics of the controlled language aimed to support engineers when writing requirements. For example, statements like "*onboard reception UV is an uplink verification stage that assesses that a telecommand has successfully reached the spacecraft using mechanism like COP-1 protocol*" includes concepts (ontology classes and individuals) like verify (action) verification type (constraint or method), verification stage, operational data item (telecommand), spacecraft, receive (action), COP-1 (constraint, protocol), that need to be modelled to represent the relationships between them. To model that relationships it is necessary to make a distinction between the actions to be completed, the agent in charge of their execution, the item on which the action is done, the

element or component that receives the output of the action and the rules that govern the execution of the action. The action is verify, it is planned to be done by the target system (or a subcomponent), and the object affected by the action is in turn another action, the reception of a telecommand at the spacecraft. The possibilities that offer OWL to represent relationships between classes and individuals by means of object properties have been applied to model these statements.

5. Conclusions

The combination of controlled vocabularies with restricted syntax to write software requirements is a promising area in those projects based on product-lines. Having an ontology that models the behaviour, constraints, data and actions of an existing system may be directly used to fill the requirements using predefined sentence patterns (filling the blanks with terms taken from the ontology). These approaches are valuable not only to ensure consistency in the specifications, but to help engineers having an overview of the main concepts managed (or to be supported) by the reused or by the target system and explore its constraints. Modelling requirements in a structured way also allows the semi-automatic verification of the specifications and improve our capabilities to generate design models from textual requirements. Improvement opportunities identified in this study are the need of user-friendly interfaces for browsing the contents of the ontology and capturing terms and the need of reviewing the number of patterns to include additional models and guide users in the selection of the most appropriates.

6. Bibliography

AMBLER, Scott. Requirements Engineering Patterns: Three approaches to motivate your developers to invest time in taking care of first things first. Dr. Dobb's , 2000. Disponible en línea: http://www.drdobbs.com/architect/184414612, Fecha acceso: 21/01/2012

BERZTISS, Alfs T. "Requirements Engineering". In: Handbook of Software Engineering & Knowledge Engineering. S. K. Chang (ed.). New Jersey etc.: World Scientific, 2002, pp. 121-143

BOYD, Stephen; ZOWGHI, Didar; GERVASI, Vincenzo. "Optimal-Constraint Lexicons for Requirements Specifications". IN: REFSQ 2007, Heidelberg: Springer-Verlag, 2007, p. 203–217 (LNCS 4542)

DURÁN TORO, A.; BERNÁRDEZ JIMÉNEZ, B.; RUÍZ CORTÉS, A.; TORO BONILLA, M. "A Requirements Elicitation Approach Based in Templates and Patterns". In: Workshop em Engenharia de Requisitos. Buenos Aires, 1999, p. 17-29

FANTECHI, A.; GNESI, S.; LAMI, G.; MACCARI, A. "Applications of linguistic techniques for use case analysis". IN: Proceedings of Requirements Engineering 2002. Society Press, 2002, p. 157-164

ISO/IEC 9126-1:2001. Software engineering -- Product quality -- Part 1: Quality model

KETABCHI, Shokoofeh; SANI, Navid Karimi; LIU, Kecheng. “A Norm-Based Approach towards Requirements Pattern”. IN: 35th IEEE Annual Computer Software and Applications Conference, 2011. DOI 10.1109/COMPSAC.2011.82

LAMSWEERDE, Axel van. Requirements Engineering: from System Goals to UML Models to Software Specifications. New Jersey: Wiley, 2009

LAMSWEERDE, Axel van. “Requirements Engineering in the Year 00: A Research Perspective”. IN: Software Engineering, 2000. Proceedings of the 2000 International Conference on 2000, ACM, 2000, p. 5-19

MAJUMDAR, D.; SENGUPTA, S.; KANJILAL, A.; BHATTACHARYA, S. “Automated Requirements Modelling with Adv-EARS”. International Journal of Information Technology Convergence and Services (IJITCS) Vol.1, No.4, 2011. DOI : 10.5121/ijitcs.2011.1406

RENAULT, Samuel; MÉNDEZ-BONILLA, Óscar; FRANCH, Xavier; QUER, Carme. “PABRE: Pattern-Based Requirements Elicitation”. IN: IEEE Proceedings of the 3rd International Conference on Research Challenges in Information Systems (RCIS). IEEE, 2009, P. 81 - 92

RENAULT, Samuel; MÉNDEZ-BONILLA, Óscar; FRANCH, Xavier; QUER, Carme. “A Pattern-Based Method for Building Requirements Documents in Call-for-Tender Processes”. International Journal of Computer Science and Applications, Vol. 6, No. 5, 2009, p. 175-202.

TJONG, Sri Fatimah; HALLAM Nasreddine; HARTLEY, Michael. “Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements” IN: Patterns. Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06) 2006

VIDEIRA Carlos; FERREIRA, David; RODRIGUES DA SILVA, Alberto. “A linguistic patterns approach for requirements specification”. Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06) 2006