Symbolic Abstraction of System Requirements

Joachim Steinmetz, Olaf Maibaum, Andreas Gerndt
Software for Space Systems and Interactive Visualization
Simulation and Software Technology, DLR
Lilienthalplatz 7, D-38108, Braunschweig, Germany

Stephan Borgert, Prof. Max Mühlhäuser
Telecooperation Group, Technische Universität Darmstadt
Hochschulstraße 10, D-64289, Darmstadt, Germany

Abstract—**Because aerospace systems become more and more complex, the pile of documents specifying the requirements for such systems grows continuously. It is obvious that it is not enough just to produce more documents. The expressiveness of system requirements has to be considered as well. However, the ambiguity of natural language generally used to define requirements impedes a manageable growth of system complexities. Therefore, new approaches of abstraction and formalization are needed.**

In this paper, we describe a semi-automatic implementation of symbolic abstraction by using methods of natural language processing and temporal logic. A graphical presentation of grammatical relations supports the comprehensibility of symbolic abstraction whereas methods defined by the Linear Temporal Logic (LTL) are incorporated to specify the temporal order of requirements. We demonstrate our approach by using exemplary requirements for the ExoMars mission.

Index Terms—**Requirements Engineering, Temporal Logic, Natural Language Processing, Model Based System Engineering**

I. Introduction

Engineering of complex systems is divided into different phases. The first phase consists of requirements engineering. It is driven by a huge set of specification documents, which are usually written in a natural language. Model-Based Systems Engineering (MBSE) allows the management of system development by introducing more abstraction techniques. For requirement specifications, abstraction schemes using symbols are recommended. The three main reasons to implement requirements using abstractions based on symbols are:

- **Management of Requirements:** Tons of documents full of system requirements are not manageable efficiently. However, if the semantics of the requirements could be expressed in a more declarative and abstract manner, computational algorithms would move the process of managing requirements of systems to a new level.
- **Ambiguity of Natural Language:** A general problem of natural languages is the ambiguous of almost all words of its vocabulary. Also sequences of words combined with grammatical rules are generally not precise. Especially in technical environments, the interpretation of the requirements can change over time and are of different meaning between involved engineering domains. For instance, the engineering process of some space systems lasts more than a decade. Therefore, a precise logic of requirements is a prerequisite of successful missions.
- **Early Verification:** Currently, the verification and validation (V&V) process is the most time consuming and cost intensive factor in systems engineering [1]. Introducing the verification process in early (and cheaper) phases of systems engineering would speedup the engineering process of consistent systems considerably. This could be achieved by symbolic requirement abstraction, which helps to identify invariant specifications of the system. Requirements can then be evaluated by formal statically verification methods.
Eventually, this leads to a formal logic which can be used to verify the system design model already in early phases of the system design (cf. Figure 1).

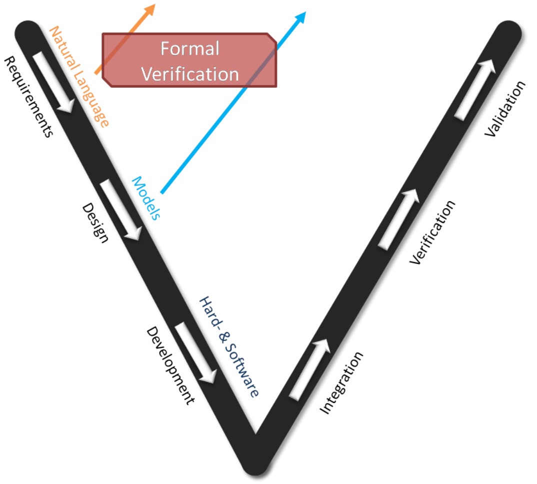

Figure 1. Early verification can reduce system development time and costs.

The symbolic logic eases the operability of requirements, avoids ambiguity of requirements, and enables early verifications. The transformation process we propose starts with a grammatical analysis, continues with a graphical representation, and eventually produces a logic structure of system requirements. The following sections illustrate the different phases of parsing, interpretation, and generation in order to receive a symbolic abstraction. In the next section, we start with a description of early phases of systems engineering. It is followed by an introduction into the diversity of requirement engineering, complemented by a section about grammatical analysis. Thereafter, the way to represent the symbolic abstraction and its grammatical relations as a graph is depicted. Finally, logical expressions are used to define the temporal logic. All descriptions are illustrated by the application on example requirements of the ExoMars project. The last section concludes the main issues and points out future work.

II. Early phases of Systems Engineering

Symbolic abstraction aims at the phase of requirement engineering where a huge set of system specifications expressed in natural language are produced. The specification process typically starts with the user who is suppose to use the system eventually in the future. She or he sketches a first system behavior as use cases. Then together with a system analyst, first requirements are noted. In several subsequent discussions, a huge set of requirements is generated.

This requirement phase is supplemented by the development of scenarios [2]. While a scenario describes a desired situation of the system operating in its environment from a functional point of view, the specification of system requirements summarizes the different capabilities and constraints of the system and the engineering process. Once defined, the requirements are the guidelines for all the work performed in the successive systems engineering phases. Nevertheless, the system developer is usually not the one who has written up the requirements. Instead of a technical definition and syntax of the system, they are more some kind of human natural expressions for the system implementation.

First designs of the technical system are made in the design phase, which starts directly after the phase of requirement engineering. The two phases have different intentions and goals. The requirements engineering process try to define invariants of the system. The system has to be complying with these invariants. Otherwise, it would breach the mission goals as defined by the first steps of user surveys. The system design phase yields a first technical sketch for implementing the system. It comes from technical system experts and not from the user who will eventually employ the system. Moreover, the design of a system is a dynamic and always amendable representation of the intended system which therefore results in permanent design modifications while engineering the system.

Although they have different intentions and goals, in real projects usually no strict boundaries between these phases exists. This is also reflected in systems engineering tools which typically do not distinct between the phases of requirements engineering and system design. One of the primary tools vendors is IBM offering a complete software suite called Rational software for design and development [3]. It contains the requirements tool DOORS and the system design tool Rhapsody. A gateway interface allows requirement interchange between both tools. But it turned out that the combined use does not cover completely the first two engineering phases with bidirectional transformations from requirement phase to system design and backward. However, IBM's tools additionally support a way of deriving test cases from use cases to enable an early execution of a testing.

To keep the original intention in mind, the approach we prefer still distinguishes between requirement engineering and system design. On the one hand, we have capabilities and constraints of the desired system and engineering process. They have been fixed by the user with the help of system analysts. These invariants describe the constant edges of the system and engineering process. On the other hand, we have the evolving system representations and implemented system elements. This system design and development object is changed over the engineering phases. At the end it will be the desired system which complies with the system requirements.

On a higher logical level, MBSE can be used to define and describe the representation of the system as models. Based on these defined system models, the system can directly be developed. The model-based engineering process supports a new methodology of designing and developing complex systems. It influences the way how to carry out the system design phase. But also the document centric requirement engineering phase is affected. In order to implement a seamless MBSE process, the requirement definition processes have to be adapted accordingly or new methods have to be developed to transform requirements after they are defined. Our proposed approach of symbolic requirement abstraction follows the second way. The engineers can still define requirements on the basis of their long term experiences without changing common work procedures and patterns. Furthermore, our approach can also be applied on already specified mission requirement documents. And vice versa, mission requirements specified with the newly proposed approaches can still be processed by older tool chains. This also helps companies, which have already large system engineering environments and a huge amount of system documentations, to overcome the misery to build more complex systems which are still manageable. With our approach, they can still use their old tools enhanced by MBSE methodology.

III. Symbolic Abstraction

To implement the desired symbolic abstraction, we need a chain of processing steps to get from natural language expressions to logical expressions. The following sub-sections examine each single step in this pipeline. First, the requirements as source are classified in more detail. Then the grammar analysis follows. The resulting hierarchical tree contains grammatical relations which are used for the next transformation step to create a graphical representation. The final step takes the structure of the graph and transforms it into logical formulas.

A. System Requirements

Requirements engineering is just one aspect of systems engineering applied in an early phase of an engineering project. In some cases, the process of requirement engineering is limited to software systems. Our approach, however, is not restricted to this special variant of systems engineering but rather our example and further work is focused on a special functional type of system requirements.

The term **requirement** comes with many different meanings. This section gives an overview about the diversity of requirements as used in systems engineering. Well known are different kinds of functional and non-functional requirements and the opposite intentions of user requirements and system requirements [4]. It is task of the system engineer to ensure the proper translation of the one into the other. In general, one can additionally distinguish between two categories of requirements. The first requirement category contains requirements which directly specify the resulting product.

Product Requirements:

1) **Input/output requirements**
2) **Structural requirements**
3) **Functional requirements**
4) **Temporal requirements**
5) **Operational requirements**
6) **Technology requirements**
7) **Environmental requirements**
    (7.1) **User requirements**
8) **Conformance requirements**

The second requirement category consists of requirements which specify the engineering process up from the design to the final product.

Engineering Requirements:

1) **Design requirements**
2) **Integration requirements**
3) **Development requirements**
4) **Verification requirements**
    (4.1) **Analysis requirements**
    (4.2) **Test requirements**
    (4.3) **Review requirements**
    (4.4) **Inspection requirements**
5) **Validation requirements**

To evaluate the formalization process from natural language requirements to abstract formulas, the following two requirements derived from the European Space Agency (ESA) space missions ExoMars [5] are used:

1) "The IMUs shall be calibrated for accuracy prior to EDL."
2) "The DMC Data Handling shall transfer to the Rover Module Data Handling the telemetry acquired during the EDL phase, prior to Rover Module egress."

These are product requirements which specify temporal order of functional behavior of the system.

B. Grammatical Analysis

Our approach of symbolic abstraction is based on requirements which are expressed in natural language. Our main focus is to handle a flat structure of text without any structure and document references which are commonly used in requirements tools. Operating on top of plain text of natural language, we are independent on the selection of a requirements tool. In this case, any requirement tool can be used to specify requirements as long as the tool can export the requirements in plain text. But the most efficient way is to integrate our techniques into existing requirement tools in order to additionally make use of their functionalities like document referencing and tracing.

For implementation of our approach, we used an unlexicalized parser for probabilistic contex-free grammars (PCFG). We did not choose a lexicalized parser because they need a lot of memory for processing long sentences. And as long as English is the used language, which is the case in several system requirements documents, the unlexicalized component parser provides near the same accurate results as a lexicalized dependency parser. For our study, the unlexicalized parser fitted perfectly. For sentences in German, a lexicalized parser is recommended [6].

For the context-free grammar, we have chosen a grammar which was generated by trainings on the New York newsletter "Wall Street Journal". This probabilistic, context-free grammar is widely used and expressed in Penn Treebank (PTB) [7] [8] notation. The grammar is provided as a serialized and compressed Java binary for loading the parser and can be used to parse English sentences. Moreover this grammar is free to use and can be extended. Besides the Java binary library, a textual version of the grammar is also available and can be compiled to the Java binary using standard Java tools. In the case of the necessity to incorporate a special technical vocabulary, e.g. to support particular engineering disciplines, we have therefore the ability to add new grammar rules or to train the grammar on this special technical corpus in order to improve the parser accuracy.

In the state the parser is loaded by using the given context-free grammar, the parser is able to work on the corpus. In our case we can process every English sentence that is recognizable by the given context-free grammar. If requirement documentations in a different language have to be processed, it is possible just to change the grammar. For instance for Chinese documents, one can switch over from the Penn English Treebank to the Penn/Colorado Chinese Treebank [9].

In those cases, also special character sets are supported. Both mentioned Treebanks and other commonly used Corpora are hosted and published by the Linguistic Data Consortium (LDC) [10].

In our current approach, only English sentences derived from requirements documents are processed. While parsing a sentence, different algorithms are executed. First of all, the parser processes each word of a sentence and creates a hierarchical tree whose nodes are annotated with Part-of-Speech (POS) tags [11] which are calculated by applying a maximum entropy model. The most common tags are noun phases (NP) an verb phases (VP). In figure 2, the yielded tree with POS tags is depicted after applying the parser on the first example requirement.

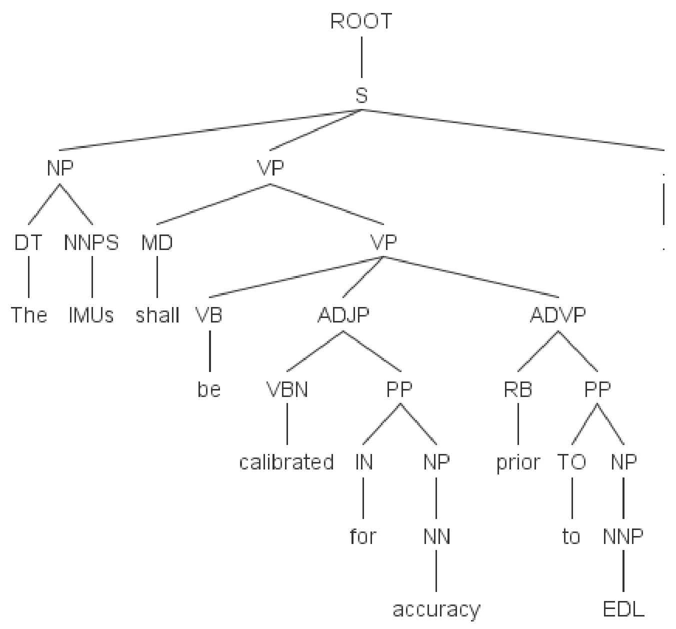

Figure 2. Hierarchical tree with POS tags after parsing the first example requirement.

Based on this tree, further grammatical analyses are applied. For analyzing the grammatical dependencies within the sentence, a hierarchically structured set of defined grammatical relations is used. Such a grammatical relation describes the relation between two nodes of the parser tree and therefore defines a grammatical dependency between two words within a sentence. The first node of this relation is called *governor* and the second node is called *dependent*. Intuitively the root of the hierarchical tree of grammatical relations is also named *dependent*. Therefore, each relation is a dependent relation if it can not be specified more precisely within the hierarchy of grammatical relations.

A grammatical relation which is a leaf within the hierarchical tree of all grammatical relations is defined by source pattern and target pattern. The source pattern is expressed by regular expressions, implemented in Java. The source pattern limits the search space to a possible set of nodes which are able to fulfill the grammatical relation. The target pattern uses Treebank Regular Expressions (Tregex) [12] to describe the local relations between first node and second node. The expression depends on the Treebank and its definition. In the case a grammatical relation is applied on a sentence, an exhaustive depth first search is executed on the parse tree. With the support of providing parent pointers in the tree structure, this bottleneck would be eliminated. As long as no indexed search on pre-indexed trees is available, it takes some time to check patterns of grammatical relations within the sentence. Phasing an English sentence yields a set of grammatical relations within the sentence which provides a basis for further symbolic abstraction.

C. Graphical Representation

The next step from natural language to formulas is the transformation of the grammatical relations into a graphical presentation. One suitable way to depict grammatical relations is to use non-cyclic, directed graphs. Graphiz [13] is an appropriate tool to visualize such graphs. It requires a special textual input format and generates appropriate diagrams. One sentence of the requirements results in one graph.

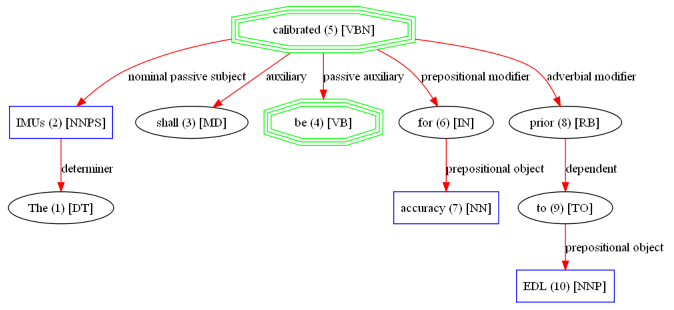

Figure 3. Graphical representation of the first example systems requirement.

A node of the graph represents one word of the sentence and an edge denotes the grammatical relation between them.

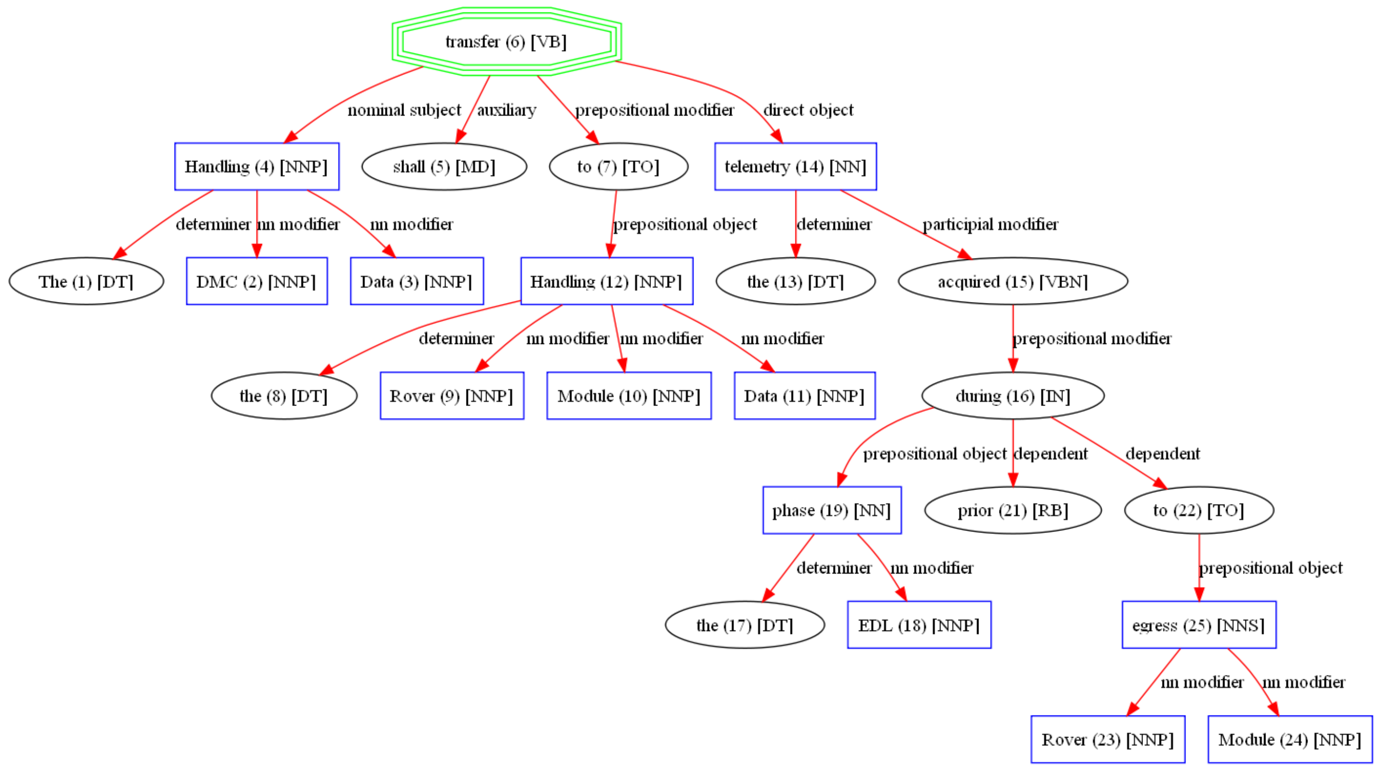

Figure 4. Graphical representation of the second example systems requirement.

An arrow starts at the *governor* and ends at the *dependent*. Labels specify the grammatical relations at the edges. At the nodes, they name the word, the index of the word in the sentence, and the POS tag. Special shapes of the nodes additionally emphasize basic grammatical elements of a sentence. A node that represents a noun is depicted as a blue rectangle. A verb in contrast is highlighted as a green octagon.

The graphical presentation of the first example system requirement is given in figure 3.

The graph represents the grammatical relations and the POS tags in an expressive way. The feature of highlighting helps to identify the core meaning of a requirement.

Figure 4 demonstrates how the graph looks like for the second example of the ExoMars requirement.

D. Logical Expression

The final step of the symbolic abstraction process is to formalize the structure of the graph by using logical formula. The proposed approach is a semi-automatic process. In order to adapt the quality of results, it permits the user to control the transformation from the structured graphical representation into logical formula. As abstraction language, the Linear Temporal Logic (LTL) [14] is used.

It provides the needed possibility to specify temporal order of system functions.

Besides propositional variables and the usual logic operators $\vee, \wedge, \neg, \Rightarrow$, the following temporal modal operators are available:

$\Diamond$	some time in the future
$\Box$	always in the future
W	always in the future *unless*
U	always in the future *until*
$\circ$	in the next state
⧫	some time in the past
■	always in the past
B	always in the past *back to*
S	always in the past *since*
$\bullet$	in the previous state

With this set of operators, it is possible to formalize the system requirements.

For the second ExoMars requirement, the temporal order of system functions is as follow: At first, the IMU (Inertial Measurement Unit) shall be calibrated [$cal()$]. Then the telemetry will be acquired [$acq()$] during the EDL (Entry, Decent, Landing) phase. Finally, this telemetry data have to be transferred [$tran()$] before the Rover module is permitted [$per()$] to discover the Mars. After the system functions are identified, the requirement can be expressed as formulas by means of the LTL.

For the two example requirements, the following formal descriptions are eventually obtained:

Requirement 1:

$$acq() \ \Rightarrow \blacklozenge cal()$$

Requirement 2:

$$\Box\neg acq() \vee \Diamond[acq() \wedge \neg per()\mathbf{U}[tran() \vee \Box\neg per()]]$$

IV. Conclusion

Technical requirements usually consist of many technical words and are written in an unstructured, human readable way. Often, many essential aspects are hidden and the temporal order can not be identified right away. The automatically generated graphical representation provides a first step to analyze the fundamental meaning of the requirement. The manual process from such a graph to a temporal logic formalizes the system requirements and eliminates inherent ambiguities.

The symbolic abstraction yields linear time logic formulas which are not as intuitive as the original requirements. However, it is not the goal to offer an alternative to the error-prone way to define requirements which can directly be used by the requirement engineers. In fact, the motivation of symbolic abstraction of system requirements is the possibility to enable early verification and further computational V&V algorithms. By the way, at least the intermediate step which automatically creates a graphical representation can be exploited for a more reliable but also intuitive approach for the analysis of system requirements.

References

[1] A. Engel, *Verification, Validation and Testing of Engineered Systems (Wiley Series in Systems Engineering and Management).* John Wiley & Sons, 2010.

[2] J. Whittle and I. H. Krüger, A methodology for scenario-based requirements capture, *Proceedings, ICSE 2004 Workshop on Scenarios and State Machines (SCESM)*, 2004.

[3] [Online]. Available: http://www.ibm.com/software/rational/

[4] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley & Sons, 2009.

[5] [Online]. Available: http://www.esa.int/esaMI/ExoMars/

[6] A. N. Rafferty and C. D. Manning, Parsing three german treebanks: lexicalized and unlexicalized baselines, *Proceedings, Workshop on Parsing German (PaGe08)*, pp. 40–46, 2008.

[7] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, Building a large annotated corpus of english: The penn treebank, *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[8] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. Macintyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, The penn treebank: Annotating predicate argument structure, *Proceedings, ARPA Human Language Technology Workshop*, pp. 114–119, 1994.

[9] N. Xue, F. Xia, F.-d. Chiou, and M. Palmer, The penn chinese treebank: Phrase structure annotation of a large corpus, *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, vol. 11, pp. 207–238, 2005.

[10] [Online]. Available: http://www.ldc.upenn.edu/

[11] K. Toutanova and C. D. Manning, Enriching the knowledge sources used in a maximum entropy part-of-speech tagger, *Proceedings, EMNLP/VLC 2000*, pp. 63–70, 2000.

[12] R. Levy and G. Andrew, Tregex and tsurgeon: tools for querying and manipulating tree data structures, *Proceedings, 5th International Conference on Language Resources and Evaluation (LREC 2006)*, 2006.

[13] E. R. Gansner and S. C. North, An open graph visualization system and its applications to software engineering, *Software - Practice and Experience*, vol. 30, pp. 1203–1233, 1999.

[14] A. Pnueli, The temporal logic of programs, *Proceedings, 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57, 1977.