Michał Grzegorzewski
Institute of Telecommunications
Warsaw University of Technology

Witold Wysota, Jacek Wytrębowicz
Institute of Computer Science
Warsaw University of Technology

Semantic Approach to Technical Requirements Specification

The paper shortly describes today practice of requirements engineering in telecommunication projects. Next it presents a method of requirements description called OpTeR (Operator Technical Requirements). We have defined ontology for this method and we have designed OpterView – an editing tool for OpTeR. The paper explains the benefits of semantic approach and gives some insights of the tool.

1 Introduction

During last 30 years, the domain of requirements engineering for software intensive project has grown and evolved. Some important standards have been accepted and more than 40 requirements management tools are available on the market today. IEEE Std 830 (1984, 1988, 1998) defines Recommended Practice for Software Requirements Specifications[8]. ANSI/IEEE Std 1471 (2000, 2007) defines Recommended Practice for Architectural Description of Software-Intensive Systems[9]. This standard was adopted by ISO/IEC JTC1/SC7 as ISO/IEC 42010:2007. The number of requirements management tools is big enough to make comparison of them a laborious task. The International Council on Systems Engineering has published such a comparison on their web pages[1]. The mentioned standards give the common understanding on requirements engineering and define recommendations for requirements processing. The tools follow the recommendations, however they differ from each other. Some of them are specialized to support particular task, others belong to big platforms that integrate tools for all the product design process.

Although requirements engineering is based on the years of practice and experience from millions of projects led all over the world in the past, there is a new idea: to define ontologies for requirements databases. This idea has been exploited already in a few university projects. We know only two of them:

- A requirements ontology has been created – *SoftWiki Ontology for Requirements Engineering* (SWORE[2]);
- Software Engineering 2.0 (that is called also SEOtology – Software Engineering Ontology)[3].

[1]http://www.incose.org/ProductsPubs/products/rmsurvey.aspx
[2]http://softwiki.de/SWORE
[3]http://www.seontology.org

We have tried to apply this idea during a project we have been working on for Telekomunikacja Polska S.A.[7]. One of our aims was to find a methodology for description of operator technical requirements dedicated for telecommunication services design in NGN–CSN[4] hybrid environment. This paper summarizes the knowledge and experiences we have collected during that project. It gives a brief description of todays practice of requirements engineering. Next it presents a method of requirements description called OpTeR (Operator Technical Requirements), which is in fact an extension of already proposed solutions[1, 3]. The important novelty, described here, is the ontology definition for OpTeR and resulting properties of the new approach. We have also designed a prototype of an editing tool (OpterView), which supports creation of semantically structured requirements specifications. The paper gives the idea how it looks like, and what is the format of registered data files.

2 Today practice of requirements engineering

The approach to requirements management (RM) differs strongly, depending on project kind and involved enterprise activity domain. RM is complex, even bureaucratic and time consuming task in safety critical or military projects. We observe an opposite approach in agile programming techniques, where RM is very simplified or almost inexistant. On the extreme, the only requirements representation is so called *user stories* – short summaries fitting on an index card explaining one aspect of what the system should do. It has been demonstrated that agile programming techniques are efficient in small and innovative projects. The telecommunication projects are usually big and expensive, due to their huge scale of deployment, thus RM should too be complex and well documented alike in the safety critical projects.

Requirements are the data of different kind, and can be categorized in different ways, e.g. [10], customer requirements, functional requirements, performance requirements, design requirements, derived requirements, allocated requirements. In the design process we start by collection and analysis of goals and intentions. Different project stakeholders expect specialized views of the requirements documentations (what is well expressed in IEEE-1471). Hence, different views are needed for business decision makers, for marketing specialists (or end-user representatives), for system architects, for programmers and for testers. In software engineering it is useful to distinguish functional and non-functional requirements, because the former can be directly implemented in software. However, non-functional requirements of a system can, in some cases, be decomposed into functional requirements for software. In other cases, a non-functional requirement may be converted into a design process requirement. For example, a system level maintainability requirement may be decomposed into restrictions on software code format or decision to build program code in pairs. As we see from the above, the requirements data arise during design, they have different levels of abstractions and granularity, and they have to be presented in form suitable for different stakeholders.

We decide to concentrate on technical requirements, because the number of them in a telecommunication project can be in the range of several thousands. Other kinds of requirement such as use case specifications or graphical interface specifications, are much less common. Moreover, technical requirements are frequently part of tender documentation, they are the base for development, for testing, and for preparation of acceptance procedures.

Requirements are written as a means for communication between the different stakeholders. Thus usually their electronic form used for exchange is PDF, RTF, HTML, or a native word processor form. Frequently they are collected using spreadsheets (e.g., Microsoft Excel). Sometimes RM tools generate them from their databases. However, our observations demonstrate that the use of word processors and spreadsheets is dominant, probably due to their ubiquitous and wide familiarity with.

Many requirements are expressed using a natural language in a more or less unstructured way.

[4]Next Generation Network - Circuit-Switched Network

Descriptions often contain weak qualifiers such as *should be* instead of stronger *must* or *has to*. This doesn't say whether fulfilling the requirement is mandatory or optional, which can lead to inconsistency in the final product. Furthermore, there is no strict template on how the document should look like. On the first glimpse such documents have something in common, but after more detailed inspection it can be seen that although the general idea is the same, they differ much in details and the similarities come from the fact that people writing the document have been using a similar document during an earlier project as a starting point (template). It's hard to call that a methodology.

Another observed thing is that usually the "specification" is written by designers accustomed to some conditions and environments, in which a system under design could be implemented and deployed, and therefore the document is biased towards some concrete solutions: hardware provider, infrastructure, protocols. Hence the document is not a requirement specification anymore but rather a documentation of in-progress implementation. Validation using such a document is not validation of the implementation against a required specification, but against the implementation itself. Thus all requirements are met automatically "by design" as they are more features of the system than real requirements.

Technical requirements are often presented in tabular form. This practice has probably been popularized by PICS (*Protocol Implementation Conformance Statement*) documents, widely used for protocol conformance tests, which has been standardized by ISO/IEC 9646-7:1995 and ETSI 300 406[5]. ORB and PixCell tools described below are examples of such approach.

The ORB (*Operator Requirements Backbone*) system was developed at Warsaw University of Technology[5] few years ago as a tool for supporting interaction between a telecom operator and its supplier. The ORB is a database system designed for creating and verification of Operator's Technical Requirements[1, 2, 3]. ORB is based on PICS proforma forms defined in ETSI standards. Using ORB it is possible for operator to create and modify requirements, and to send this specification to a supplier who can declare if they are implemented in his product(s). The whole process is divided into five separate phases (creation, parametrization, declaration, verification, archiving). In the first phase operator creates general requirements for system under design like requirements hierarchy and structure, validation roles, etc. Then there is parametrization process in which all specific informations are defined. In the third phase requirements are sent to supplier who declares implementation status of defined requirements. After this the verification process is performed by operator who checks the real implementations status (for example by testing process). Finally all documentations are moved to an archive for future use.

The main reason why the ORB system is not in use widely is the fact that all requirements' specifications in ORB system are based *only* on PICS forms which in fact is not enough for today's services. The next step in development of tools for creating and using operator's requirements was PixCell.

PixCell is a small tool based on MS Excel which was used to present requirements for telecom services. This is also a university tool developed in the Institute of Telecommunications at Warsaw University of Technology[6].

The main goal of this tool is to extend PICS tables and allow users to decide about how detailed requirements they want to work with. In PixCell there are internal links, which can be used to see relations between requirements and check if changes in one row have impact on other parts of the document. It also provides some simple algorithms for calculating current implementation status and checking its validity. Although this tool is in prototype stage the experience gathered with it has been useful during work on OpTeR methodology.

A tabular description is less ambiguous, compared to a plain text definition in a natural language. Use of spreadsheets also has some disadvantages. Although the meaning of data can be distinguished thanks to column names, the type of data inserted into a cell can vary (e.g., it can be a character string

[5]Institute of Telecommunications, Laboratory of Testing and Verification

or an integer value). Moreover a user can split the cell vertically or horizontally which makes the exported files very difficult to process automatically.

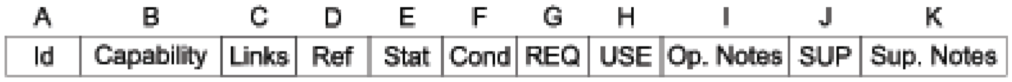
Regardless of technical requirements record contents, traceability of the specification has to be assured. Typical fields we provide for it are: *Author*, *Creation Date and Time*, *Modification Date and Time*, *Origin*, *State of progress*. Traceability data can be attributed to a requirements document, to selected branches in a hierarchy tree of requirement records, or even to every requirement record. Filling traceability data is boring, presenting them for most design tasks is useless, thus designers creating requirement specifications with word processors and spreadsheets frequently ignore them. For this reason and to eliminate the above mentioned disadvantages of spreadsheet use, it is recommended to work under requirement specifications with specialized computer aided tools.

3 OpTeR methodology

After a lot of analysis and taking into account limitations in tools described above, a new methodology was created in Institute of Telecommunications at Warsaw University of Technology. The methodology called OpTeR[6] (described in detail in [7] and [6]) was designed for describing and working with telecom operator's requirements. However many other complex ICT projects can use OpTeR, not only those led by an operator. OpTeR consists of two parts:

- formal description of requirements represented as a table including references to external sources of information (i.e., external standards that requirements are linked to);
- specification that is not directly written in requirements' table but has to be taken into account and notes that can not be formalized.

The nature of all these parts is different. Formal description (the main part of requirement definition) includes all requirements and are written in a tabular form. There is no space for misunderstanding or misinterpretation. In specification part there are definitions of terms and explanations given in a natural language without words such as *must*, *should*, *may*, etc. References to internal operator's regulations concerning implementation of the system under design can also be placed there. The crucial part of the methodology is the formal description part. The figure 1 shows the structure of OpTeR requirements table.



A	B	C	D	E	F	G	H	I	J	K
Id	Capability	Links	Ref	Stat	Cond	REQ	USE	Op. Notes	SUP	Sup. Notes

Figure 1: OpTeR table structure

Every row in the table describes one requirement. It is possible to organize the requirements in a tree hierarchy. Requirement identifiers allow to distinguish the position in the hierarchy. The higher and lower level requirements are described in the same way. Below there is a detailed description of every field in the requirement record structure:

(A) Id (identifier) – this is the identifier of one single requirement. It must be unique in the whole requirements document and should express hierarchy level (for example using the "x.y.z" format, where "y" is a more general requirement then "z" and less than "x").

(B) Capability – in this field there is the title of general group of requirements or description of the one singe requirement with short explanation. There is no need to write detailed description, but it is recommended that the name would be unique in whole table.

[6]Operator Technical Requirements

(C) Links – in this field some identifiers can be placed to designate connections between requirements. They are static, without any description of the link's character. In most situations in this field we can put connections to requirements which are used to fulfil the functionality described by the current item.

(D) References – in this field we can add references to external documents related to the capability being described. We can identify two general sources of such documents:

- national or international standards that we (as an operator or supplier) are obligated to respect;
- other requirement documents that are useed in our environment (company, group, etc.).

There is no limit to number of references we can make, the more references we put here the less time we spend later to search for information about the capability. Sometimes it is useful to add a link to a PICS document which contains descriptions of external constraints.

(E) Stat (standard status) – this field is used to show what is the referenced standard status (an external standard or other document like PICS tables or national regulation). We have to be complaint with external constraints and to be aware if they are required or optional.

(F) Condition – this field is used to select a value from the Req column (described below). The condition is a logical statement which can evaluate to *true* or *false*. The variables are taken from SUP field. Depending on the condition, the first value or the second value (after keyword "else") from the Req field is selected as binding. If there is no second value we assume that the requirement status is "n/a" (not applicable).

(G) Requirement – this field is used to show the implementation status from operator/designer point of view. This status shows if the capability is mandatory, optional or prohibited to implement by the vendor. Two values separated by "else" can be given in this field. The selection is made based on the condition described above.

(H) Use – this part of requirement describes if we are going to use this capability or not. In some design scenarios this field can be useful, e.g., when we plan subsequent versions of the system under design (*we don't use the feature now but we may use it in future*).

(I) Op. Notes (Operator's notes) – This field is designed for operator's/designer's comments however it is recommended to avoid using it, because it is an informal way of describing requirements and can lead to same misunderstandings.

(J) Sup (support) – This field is designed for vendor who can here confirm the implementation status of the feature (if it is implemented or not). It is recommended that all requirements would be defined strictly enough so that implementation status is easy to determine without any comments.

(K) Sup. Notes – in this field the vendor can add additional comments.

Additionally the described structure of OpTeR can be presented in different way (i.e., by hiding selected columns) depending on user's role and current project life cycle phase. It is possible – during contacts with suppliers – that column USE and/or Req are hidden in order to avoid suggesting expected answers. The operator does not show what functionality is required so the supplier can fill the table according to his current implementation status. For better understanding of OpTeR methodology the OpTeR ontology was created and described in following section.

4 OpTeR ontology

Semantic networks is a formal way of storing knowledge in a form that can be understood by machines. The data is represented by triplets [subject, predicate, object] that allows for knowledge discovery and formal verification and validation of a model through inferring. The essence of the knowledge is kept without determining the way it is going to be presented to the receiver[12, 4].

As we decided to base our project platform on semantic networks, it became natural we needed a semantic approach to managing operator technical requirements as well. A consequence of this was to create an ontology for the OpTeR tabular notation.

4.1 OpTeR semantic description

To define the ontology for OpTeR we have used OWL[7]. This language creates descriptions according to the RDF[8] syntax, which makes our ontology compatible with most popular tools in this domain. The core of our semantic description is a set of classes that define terms that can be mapped to contents of the OpTeR table.

The first entity to be created was the `ReferenceDocument` class. It contains objects that are external documents that can be referenced from within a requirement through the `references` property.

The second class is `Usage` that holds in its instances the information about a particular requirement being in use or not. This class has three instances – `UseYes`, `UseNo` and `UseNotApplica-ble`. They are possible values of the `uses` property, which is a relation between a feature (requirement) and its usage state.

Each feature can be supported by the provider, not supported or partially supported. This information is kept by the `SupportValue` class that has the following instances: `SupYes`, `SupNo`, `SupPartly` and `SupNotApplicable`. The latter is a special value used if the support for a particular requirement is not determined.

The next class is `RequirementValue` that contains instances, which describe the requirement state of each feature through the `requires` property.

The final class – `Row` is a domain for the properties described earlier as well as other properties we will briefly describe. An instance of this class represents a single row in the document. Its properties contain values for subsequent columns of the OpTeR table associated with a particular requirement.

Not all of the OpTeR structure are covered by the listed properties. There are also three datatype properties, which map not to objects but to values of well known simple types – `capability`, `operatorNotes` and `supplierNotes` that contain textual descriptions from appropriate columns of the OpTeR table.

There are also four special properties that form relations between rows. The properties can be matched in pairs where one property in pair is the inverse of the other one. OpTeR specifications are not flat tables but rather trees that are only displayed as tables. The first of the mentioned pairs is responsible for groupping rows into hierarchies. It can match a row to a list of its lower level rows (children) through the `hasSubrow` property. Similar the `isSubrowOf` property allows to find the parent of each row, if it exists. The second pair (`linksTo` and `isLinkedTo`) creates bonds "across" rows – it represents the functionality to link rows related to each other in a many-to-many cardinality.

Table 1 presents all the properties of the `Row` class. One can notice that this matches the structure of the OpTeR table presented earlier. The properties constitute the verbs (predicates) and the classes constitute subjects and objects of the mentioned tripplets. Thanks to that we can even form sentences

[7]Web Ontology Language

[8]Resource Description Format

in natural language that can be mapped directly to the specification (i.e., *Row #1.1 is subrow of row #1*, *Row #7.6 is linked to row #2.8.1*) and still be perfectly understood by both the human and the machine.

Table 1: Properties of the `Row` class

Name	**Range**	**Cardinality**	**Comment**
`isSubrowOf`	Row	$0-1$	*points to the parent row*
`hasSubrow`	Row	∞	*points to child rows*
`capability`	string	$0-1$	*the name of the requirement*
`linksTo`	Row	∞	*list of rows linked to the current one*
`isLinkedTo`	Row	∞	*list of row the current one is linked to*
`references`	ReferenceDocument	∞	*list of external documents*
`condition`	LogicalExpression	$0-1$	*logical expression*
`requires`	Requirement	∞	*require state of a feature*
`uses`	Usage	$0-1$	*whether the requirement is used by the service*
`operatorNotes`	string	∞	*remarks from the operator*
`supported`	SupportValue	$0-1$	*whether the provider supports the requirement*
`supplierNotes`	string	∞	*remarks from the supplier*

4.2 Semantic requirements

Below there is a detailed description of possible semantic usage to build a table of requirements in OpTeR methodology.

In column “Support” it is allowed to use status:

- Y (yes) – the capability is fully implemented
- N (no) – the capability is not implemented
- P (partly) – this functionality is supported with some remarks. The “P” status is allowed but it should be avoided due to futures problems or misunderstandings.

Possible usage of column “USE”:

- Y (yes) – this capability is used by owner of the requirements
- N (no) – this capability is currently not used
- N/A (not applicable) – for this capability this column is not used.

Possible values for column Req (Requirement):

- M (mandatory) – this functionality *must* be implemented
- O (optional) – it is up to vendor to implement or no this capability
- O.i (optional with limitation) – it is required to implement at least *i* capabilities marked like this. For example if there is a group of four requirements mark like *O.1* it is needed to implement only *one* of the requirement. There rest three should not be implemented.
- X (eXcluded) – it is *not allowed* to implement this functionality
- NR (not required) – owner of this document is not interested if this capability is implemented or not

– N/A (Not Applicable) – the status of this capability is unknown. Value of the column Req can be defined as a result of condition.

There are lots of possible combinations of the described columns. Each of this combination represents a different situation, some of them are prohibited (because of logical conflict or noncompliance with standards) and some of them are used for making the requirements weaker or stronger. More information can be found in [3], [7] or [6]. For better understanding and easier use of OpTeR ontology the *OpTeR View* tool was implemented and is described in next section of this paper.

5 OpTeR View editing tool

OpterView is a prototype of a graphical editor for OpTeR. Here we describe its purpose and give some details about its architecture and functionality. Finally we show possible enhancements of the tool.

The application was created as an example of a view compliant to the architecture of a system for managing resources we were developing at that time. We also needed a tool to quickly create ontologies according to the OpTeR specification so that we could use them in other parts of the project.

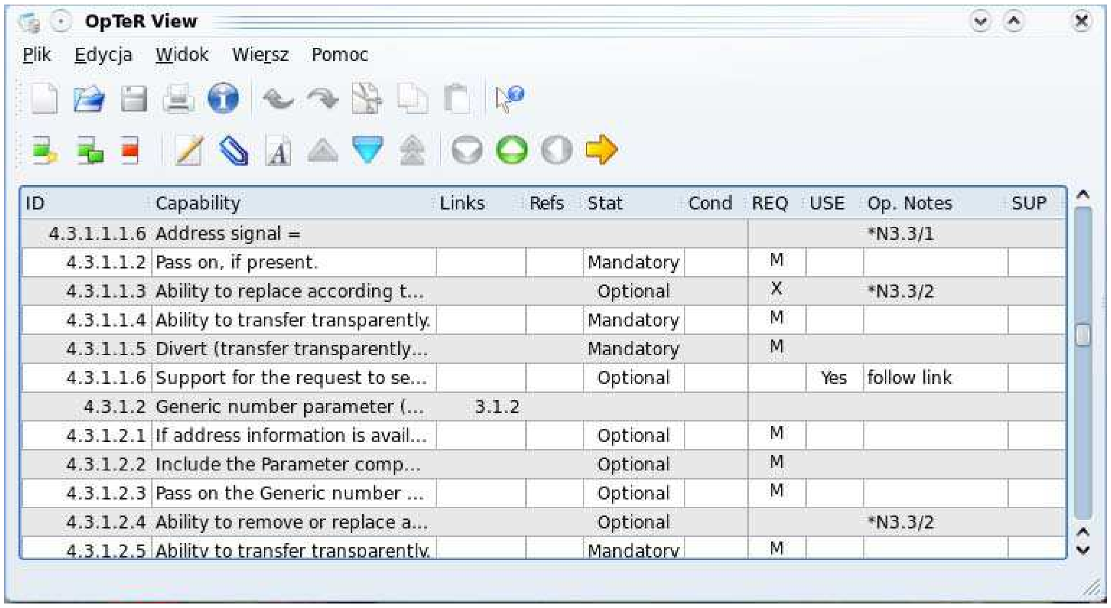
The purpose of the editor is to provide an environment for creating OpTeR descriptions in a way that is more convenient than using a simple spreadsheet. OpterView stores its data in a form of OpTeR ontology, which makes it possible to use the generated description with other semantic-enabled tools. At the same time it doesn't require any skills or knowledge related to semantic networks – the user only sees a table, which contains rows and columns according to what is specified in OpTeR ontology.

The tool was created in Qt4 – a cross-platform toolkit and application framework. Use of a portable environment makes it possible to deploy the application on almost any modern system (including Windows, Unix, MacOSX and even mobile devices). Portability is obtained at source code level, thus the program is compiled into native code of each platform resulting in a fast system with low memory footprint and no dependencies other than the framework itself. Qt4 provides a very efficient and simple mechanism for internationalization of applications – the approach is used by OpterView and allows to run the application in any language that can be expressed with Unicode (including right-to-left written languages). This all makes it, that OpterView integrates well with the look and feel of other applications for the platform much unlike heavy applications based on Java. Figure 2 shows a screen shot of OpterView displaying an OpTeR specification of the ISUP protocol.

Currently OpterView is in a prototype stage. Its functionality is limited but it allows for creation of skeletons or even full OpTeR tables. There are still some quirks and instabilities in the application's performance but the main functionality like creating, loading and saving the specifications is ready.

What doesn't currently exist in OpterView is the support for different aids, making use of the semantic nature of the editor as well as its processing power to implement functionality such as doing some checks in the background while the user is thinking about his next steps or otherwise interacting with the program. One of the easiest yet very useful features that can be added is full detection of cycles in chains of links between rows in the OpTeR table – the current version of the software is able to detect simple cycles only. Semantic networks are a formal method and thus give us the apparatus to mathematically prove correctness or contradiction of the specification. The tool should be able to cooperate with a semantic reasoning engine like Pellet or Fact++[11, 13] or alternatively provide basic reasoning support itself.

The future of the editor is closely related to the future of OpTeR methodology. If the notation gains popularity, then OpterView seems the natural choice for a dedicated application to operate on OpTeR tables. In such situation the development of the program could be resumed and the missing functionality could be added.



ID	Capability	Links	Refs	Stat	Cond	REQ	USE	Op. Notes	SUP
4.3.1.1.1.6	Address signal =							*N3.3/1	
4.3.1.1.2	Pass on, if present.			Mandatory		M			
4.3.1.1.3	Ability to replace according t...			Optional		X		*N3.3/2	
4.3.1.1.4	Ability to transfer transparently.			Mandatory		M			
4.3.1.1.5	Divert (transfer transparently...			Mandatory		M			
4.3.1.1.6	Support for the request to se...			Optional			Yes	follow link	
4.3.1.2	Generic number parameter (...	3.1.2							
4.3.1.2.1	If address information is avail...			Optional		M			
4.3.1.2.2	Include the Parameter comp...			Optional		M			
4.3.1.2.3	Pass on the Generic number ...			Optional		M			
4.3.1.2.4	Ability to remove or replace a...			Optional				*N3.3/2	
4.3.1.2.5	Ability to transfer transparently.			Mandatory		M			

Figure 2: OpterView

6 Conclusions

The paper shortly describes the OpTeR methodology for technical requirements specification and demonstrates the way of describing requirements in a semantic way. It also introduces the OpterView editor which has been created as a prototype to support OpTeR.

If we assumed that only humans process requirements then building ontology for them would be superfluous. However, if we intend to process them by computer tools, then semantic data will be indispensable. When the number of technical requirements is larger than several hundred, working with them in a paper form is inefficient. It starts to be like with programming code – today nobody prints it nor analyses paper listings, we have computer tools for that. It is obvious that we need such tools like OpTeR, and that they should have interfaces to communicate (or exchange data) with other computer tools.

We do not pretend that OpTeR is the best method for technical requirements specifications. We state that it can be useful. It is probable that designers use several specification methods and related tools for big projects. In that case, the possibility of mutual data exchange between the tools is really desired. If the data collected has semantic description then it will be easy to build links between different tools for requirement specification and processing.

The presented OpterView editor helps to collect requirement data, probably no so flexibly as it could be done with MS Excel, but the use of OpterView is much simpler than the use of Excel and what is more important – the resulting files contain data with semantic description. It is easy to create a tool, which could generate specialized views of collected requirements. Such views could be conceived to help reasoning by different specialists about the requirements.

We believe that specialized and lightweight tools which can cooperate exchanging semantically recognizable data will be dominant in future requirement engineering. New projects are inherently different from the past projects and they reveal new problems. Huge CASE tools that try to solve all problems already start to be inefficient and difficult to use. Hence probably presented here semantic OpTeR methodology shows a good direction for future requirement engineering tools.

References

[1] Krzysztof M. Brzeziński. Metodyka tworzenia i zapisu wymagań operatora dla implementacji protokołu telekomunikacyjnego. In *PWT'2003, VIII Poznańskie Warsztaty Telekomunikacyjne*, 2003.

[2] Krzysztof M. Brzeziński, R. Artych, and D. Mastalerz. Komputerowe wspomaganie stosowania wymagań technicznych dla protokołów. In *Srodowisko wspomagajace tworzenie i eksploatacje Wymagan Operatora dla implementacji protokolow sygnalizacyjnych, dokumentacja uzytkowana systemy ORB v.1.0*.

[3] Krzysztof M. Brzeziński and Michał Grzegorzewski. Komputerowe wspomaganie stosowania wymagań technicznych dla protokołów. In *materiały Krajowego Sympozjum Telekomunikacji KST'04, tom. B*, pages 177–186. Instytut Telekomunikacji Politechniki Warszawskiej, 2004.

[4] Allan M. Collins and Elizabeth F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, November 1975.

[5] ETSI. ETS methods for testing and specification (mts); protocol and profile conformance testing specifications; standardization methodology. *ETSI Std 300 406*, 1995.

[6] Michał Grzegorzewski. Komputerowe wspomaganie procesów stosowania wymagań technicznych dla protokołów sygnalizacyjnych. In *Master Thesis*. Instytut Telekomunikacji Politechniki Warszawskiej, 2004.

[7] Michał Grzegorzewski, Kamil Karwowski, Witold Wysota, and Jacek Wytrębowicz. Metodyka opisu struktur, usług i protokołów hybrydowego środowiska NGN/CSN dla wspomagania działań Operatora. Technical report ordered by R&D Dept. of Telekomunikacja Polska S.A. Serial number: 26/06 (501/E/1036/4470), October 2008.

[8] IEEE. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, Oct 1998.

[9] IEEE. Systems and software engineering - recommended practice for architectural description of software-intensive systems. *ISO/IEC 42010 IEEE Std 1471-2000 First edition. 2007-07-15*, pages c1–24, 15 2007.

[10] Defense Acquisition University (Producer). Systems engineering fundamentals. Defense Acquisition University Online Publication Resources, 2001.

[11] E Sirin, B Parsia, BC Grau, A Kalyanpur, and Y Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, June 2007.

[12] John Sowa, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge (Morgan Kaufmann Series in Representation and Reasoning)*. Morgan Kaufmann Pub, May 1991.

[13] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.