Vilnius University
INSTITUTE OF MATHEMATICS AND INFORMATICS
L I T H U A N I A

INFORMATICS ENGINEERING (07 T)

THE RESEARCH ON MODEL TRANSFORMATIONS, BASED ON DOMAIN METAMODEL, FOR DESIGNING REQUIREMENTS SPECIFICATIONS

Neringa Makrickienė

October 2017

Technical Report MII-DS-07T-17-10

VU Institute of Mathematics and Informatics, Akademijos str. 4, Vilnius LT-08663, Lithuania

www.mii.lt

Abstract

In recent year ontologies – shared conceptualizations of some domain – are increasingly seen as the key to further automation of information processing. There are many applications of such an approach, e.g. automated information processing, information integration or knowledge management, to name just a few.

Although many approaches for representing and applying ontologies have already been devised, they still haven't found their way into enterprise applications.

Ontologies, for software design and development, can be used with the following objectives [29] [26]:

• Specification: ontologies are used to specify either the requirements and components definitions (informal use) or the system´s functionality.

• Confidence: ontologies are used to check the system´s design.

• Reusability: ontologies could be organized in modules to define domains, subdomains and their related tasks, which could be later reused and/or adapted to other problems.

• Search: ontologies are used as information repositories.

• Reliability: ontologies could be used in (semi)–automatic consistency checking.

• Maintenance: ontologies improve documentation use and storage for system's maintenance.

• Knowledge acquisition: ontologies could be used as a guide for the knowledge acquisition process. Within Software Engineering, two main roles for ontologies have been considered [30]:

• Ontologies for the Software Engineering Process: the definition, re–use and integration of software components is aided by the use of ontologies as the conceptual basis.

• Ontologies for the Software Engineering Domain: the use of ontologies to describe the structure and terminology of the software engineering domain itself.

These are the key reasons why ontology is included in this research to represent knowledge base. The main considerations in the research will be requirements specification generation from ontology. Further work will include the research on ontologies, their languages and tools, ontology based model transformations and requirements engineering.

Problem statement

The study of an information system requirements should result in the establishment of well-defined functionalities and attributes agreed by the stakeholders. If the functionalities are defined as incomplete or incorrect, the software may not meet the expectations of users. Factors that could lead to an inadequate process of requirements elicitation can be [37]:

- *Ambiguous Requirements*: which produce lost of time and repeated work. Their origin resides in the diverse stakeholders, who produce different interpretations of the same requirement. Moreover, one stakeholder can interpret the same requirement in diverse ways. The ambiguity conduces to mistaken product tests.
- *Insufficient Specifications*: they produce the absence of key requirements. This conduces to developers' frustration, because they base their work in incorrect suppositions and, so, the required product is not developed, which displeases the clients.
- *Requirements not completely defined*: they make impossible the project secure planning and its monitoring. The poor requirements understanding leads to optimistic estimations, which return against when the agreed limits are surpassed.
- *Dynamic and changing requirements*: which require constant requirements revision in order to help to understand new clients needs and to identify how they can be satisfied.

In order to reduce the negative effects of the previous factors on the RE processes, the ontologies can be used. The potential uses of ontologies in RE include the representation of:

- The requirements model, imposing and enabling a particular paradigmatic way of structuring requirements;
- Acquisition structures for domain knowledge;
- The knowledge of the application domain [37].

An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements so difficult [38].

Object

The object of the research is a method on designing requirements specifications, based on domain metamodel.

Goal and objectives

The main goal of the research is to propose the method for designing requirements specification, in order to improve existing methods and to meet a standard criteria of a good system requirements specification.

To meet this goal, several objectives arise:

1) To investigate the existing situation of requirements specification design processes;
2) To make a research on using domain methodologies in requirements engineering;
3) To present an improved methodology on designing requirements specification that meets formal criteria;
4) To evaluate proposed method by conducting experiments based on the chosen domain;
5) To propose results on the research.

Research methodology

Literature review, comparative analysis.

Results approval

1. Veitaitė I., Lopata A., Žemaitytė N. (2016) Enterprise Model based UML Interaction Overview Model Generation Proces. 19th International Conference on Business Information Systems, BIS2016 International Workshop, Series: Lecture Notes in Business Information Processing. ISBN 978-3-319-26762-3.

Contents

1. ANALYSIS ON DOMAIN METAMODELS PRACTISE, IN A CASE OF SOFTWARE REQUIREMENTS SPECIFICATIONS MODELLING....7
2 Other activities during 2015-2017 year of study....54
3 References....55
4 Appendixes....60
1 Introduction....61
2 Enterprise Modelling and Ontologies relation....61
3 Transformation Algorithm....62
4 UML Interaction Overview Model Transformation....64
5 Conclusions....66
6 References....67

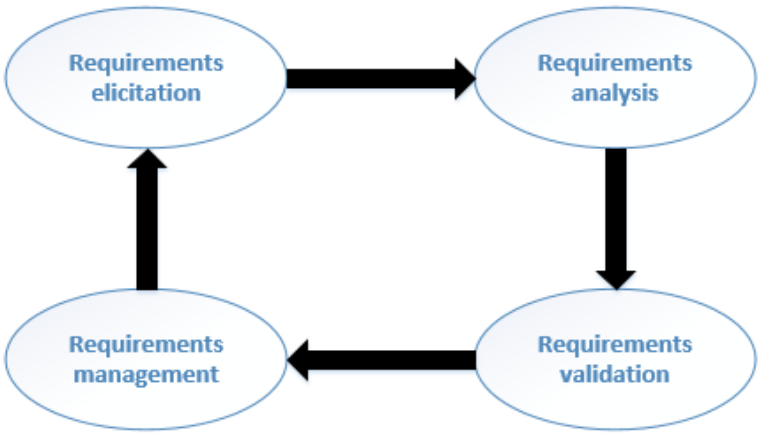
1. ANALYSIS ON DOMAIN METAMODELS PRACTISE, IN A CASE OF SOFTWARE REQUIREMENTS SPECIFICATIONS MODELLING

1.1 The concept of requirements engineering

A requirement is a statement that identifies a product or processes operational, functional, or design characteristic or constraint, which is unambiguous, testable, or measurable and necessary for product or process acceptability (ISO 2007).

Requirements Engineering (RE) is concerned with the elicitation, evaluation, specification, consolidation, and change of objectives, requirements, functionalities, qualities, and constraints to be achieved by a software-intensive system. RE has the objective to establish a complete, consistent and unambiguous description of requirements (Requirements Specification) for a given application domain on an abstract conceptual level. This incremental process involves stakeholders from different backgrounds and requirements engineers.

Requirements Engineering is the branch of Systems Engineering concerned with the development of requirements through a systematic, iterative and co-operative process. This process includes the elicitation, negotiation, specification and validation of requirements. It is also concerned with the relationship of these RE artefacts to precise specifications of software behaviour, their evolution over time and across software families. Requirements and all related artefacts are documented in a Requirements Specification that needs to be validated regarding customer wishes, correct understanding and accuracy [40].



Reference: [40]

Fig. 1 Requirements Engineering Activities

Simplified process of gathering requirements is presented in Fig. above. The process is presented as a circle, as every step can be repeated, due to problems in every stage. The most important ones are Requirements elicitation and Requirements analysis. In some cases these two stages are paired together, because they deeply depend on each other. Problems in requirements elicititation could be misunderstanding between client and system analyst, wrongly expressed and interpreted goals, processes, domain knowledge. This leads to wrong requirements analysis process (or not neccesserilly). Also, due to lack of knowledge and experience, process can trigger problems in analysis stage, even elicitation was successful. This could be different stakeholders produce different interpretations for the same requirement at this stage. And repeated work is needed then. During requirements analysis and validation, requirements specification document is prepared. In most cases, during analysis, because validation is verifying with a customer if requirements meets their needs. In this research we will focus on requirements analysis stage and how to improve it. But by improving this stage, other stages should generate better results as well, as it is very connected to each other, as we see in figure above.

Even the process shown is simplified, this does not mean, the process is simple. According to SEBoK it includes more action points. Major activities and tasks during this process include [77]:

- Analyzing the stakeholder requirements to check completeness of expected services and operational scenarios, conditions, operational modes, and constraints.

- Defining the system requirements and their rationale.
- Classifying the system requirements using suggested SEBoK classifications.
- Incorporating the derived requirements (coming from architecture and design) into the system requirements baseline.
- Establishing the upward traceability with the stakeholder needs and requirements.
- Establishing bi-directional traceability between requirements at adjacent levels of the system hierarchy.
- Verifying the quality and completeness of each system requirement and the consistency of the set of system requirements.
- Validating the content and relevance of each system requirement against the set of stakeholder requirements.
- Identifying potential risks (or threats and hazards) that could be generated by the system requirements.
- Synthesizing, recording, and managing the system requirements and potential associated risks.
- Upon approval of the requirements, establishing control baselines along with the other system definition elements in conjunction with established configuration management practices.

A Software Requirements Specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.

To the customers, suppliers, and other individuals, a good SRS should provide several specific benefits, such as the following [IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830-1998, 1998.]:

Establish the basis for agreement between the customers and the suppliers on what the software product is to do. The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs.

Reduce the development effort. The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the

requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

Provide a basis for estimating costs and schedules. The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.

Provide a baseline for validation and verification. Organizations can develop their validation and verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.

Facilitate transfer. The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.

Serve as a basis for enhancement. Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

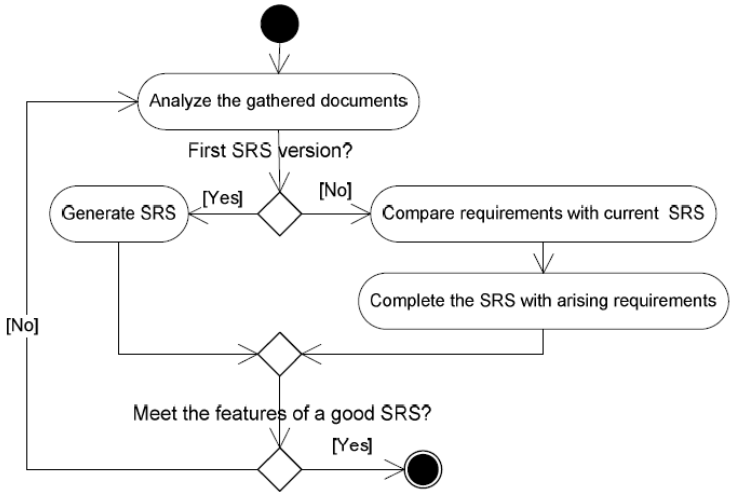
Table 1. Characteristics of good SRS

Criteria	Description
Correct	An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet. There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior specification, such as a system requirements specification, with other project documentation, and with other applicable standards, to ensure that it agrees. Alternatively the customer or user can determine if the SRS correctly reflects the actual needs. Traceability makes this procedure easier and less prone to error.
Unambiguous	An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term. In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.
Complete	An SRS is complete if, and only if, it includes the following elements: All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system speci- fication should be acknowledged and treated. Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values. Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

Consistent	Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.
Ranked for importance and/or stability	An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement. Typically, all of the requirements that relate to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable. Each requirement in the SRS should be identified to make these differences clear and explicit. Identifying the requirements in the following manner helps: - Have customers give more careful consideration to each requirement, which often clarifies any hidden assumptions they may have. - Have developers make correct design decisions and devote appropriate levels of effort to the different parts of the software product.
Verifiable	An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.
Modifiable	An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to: - Have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross referencing; - Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS); - Express each requirement separately, rather than intermixed with other requirements.
Traceable	An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended: - Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents. - Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.

Šaltinis: IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830-1998, 1998.

SRS generation can be described as a process also.



Reference: [40].

Fig. 2 SRS generation process

The quality of SRS also is a repeatable process where competency questions written in natural language are interpreted.

All of the system analyst would like to write requirements specifications meeting these requirements. But it very depends on the experience of the system analyst, so human factor is playing a key role while preparing a specification.

On the other hand, nobody writes them from scratch, unless it is very informal requirements or has a little scope. But mostly of the proffesionals in this field use templates. Or company know-how. Which could be a template also. There are many templates created and can be found online. But to be closer to perfection, formal templates are used, and they are only a few known in the world:

- **Volere** Requirements Specification Template, copyright © 1995 – 2007 the Atlantic Systems Guild Limited;
- **IEEE** template 830;
- **IBM** template;
- **ISO** standard.

Volere Requirements Specification Template is intended for use as a basis for requirements specifications. The template provides sections for each of the requirements types appropriate to today's software systems. It can be downloaded as a pdf version from the Volere site and adapted it to requirements gathering process and requirements tool. The Volere site also has a Word RTF version. The template can be used with Requisite, DOORS, Caliber RM, IRqA, Magic Draw and other popular tools.

IEEE template is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide (IEEE).

IBM template provides access to the detailed system requirements information on the supported releases of IBM Business Process Manager Standard (IBM).

Templates are convenient to use, but it does not provide reasoning tools, also it gives a structure and tips/tricks for writing requirements, but it does not provide any information about a domain. For this reason ontologies should be used to accompany templates for requirements specification preparing process.

ISO standard. It is not a template, but very important standard which companies should keep while creating software. ISO/IEC/IEEE 29148:2011 contains provisions for the processes and products related to the engineering of requirements for systems and software products and services throughout the life cycle. It defines the construct of a good requirement, provides attributes and characteristics of requirements, and discusses the iterative and recursive application of requirements processes throughout the life cycle. ISO/IEC/IEEE 29148:2011 provides additional guidance in the application of requirements engineering and management processes for requirements-related activities in ISO/IEC 12207 and ISO/IEC 15288. Information items applicable to the engineering of requirements and their content are defined. The content of ISO/IEC/IEEE 29148:2011 can be added to the existing set of requirements-related life cycle processes defined by ISO/IEC 12207 or ISO/IEC 15288, or can be used independently. ISO/IEC 12207:2008 (IEEE Std 12207-2008) and ISO/IEC/IEEE 15288:2015, - specifies the required information items that are to be produced through the implementation of the requirements processes,- specifies the required contents of the required information items, and- gives guidelines for the format of the required and related information items (ISO).

There are several pitfalls that will inhibit the generation and management of an optimal set of system requirements, as discussed in Table below.

Table 2. Major Pitfalls with Definition of System Requirements

Pitfall	Description
Insufficient Analysis of Stakeholder Requirements	If the receivers of the stakeholder requirements do not perform a sufficient critical analysis of them, the consequence could be difficulties translating them into system requirements and the obligation to come back to the stakeholders, losing time.

Insufficient Analysis of Operational Modes and Scenarios	The operational modes and operational scenarios are not sufficiently analyzed or defined by the person in charge of writing the system requirements. Those elements allow the structuring of the system and its use early in the engineering process and help the designer to remember functions and interfaces.
Incomplete Set of System Requirements	If the system requirements are not sufficiently precise and complete, there is a great risk that the design will not have the expected level of quality and that the verification and validation of the system will be delayed.
Lack of Verification Method	Delaying the capture of verification methods and events for each system requirement; identification of the verification approach for each requirement often provides additional insight as to the correctness and necessity of the requirement itself.
Missing traceability	Incorrect or missing traceability of each requirement, both to an upper-level "parent" requirement as well as allocation to an inappropriate system or system element.

Reference: [77]

To overcome pitfalls and to develop a successful requirements specification, widely known methodologies can be adjusted.

1.1.1 Ontologies in RE approach

The study of an information system requirements should result in the establishment of well-defined functionalities and attributes agreed by the stakeholders. If the functionalities are defined as incomplete or incorrect, the software may not meet the expectations of users. Factors that could lead to an inadequate process of requirements elicitation can be [37]:

- *Ambiguous Requirements*: which produce lost of time and repeated work. Their origin resides in the diverse stakeholders, who produce different interpretations of the same requirement. Moreover, one stakeholder can interpret the same requirement in diverse ways. The ambiguity conduces to mistaken product tests.
- *Insufficient Specifications*: they produce the absence of key requirements. This conduces to developers' frustration, because they base their work in incorrect suppositions and, so, the required product is not developed, which displeases the clients.
- *Requirements not completely defined*: they make impossible the project secure planning and its monitoring. The poor requirements understanding leads to optimistic estimations, which return against when the agreed limits are surpassed.
- *Dynamic and changing requirements*: which require constant requirements revision in order to help to understand new clients needs and to identify how they can be satisfied.

In order to reduce the negative effects of the previous factors on the RE processes, the ontologies can be used. The potential uses of ontologies in RE include the representation of:

- The requirements model, imposing and enabling a particular paradigmatic way of structuring requirements;
- Acquisition structures for domain knowledge;
- The knowledge of the application domain [37].

An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements so difficult [38].

Simplified, ontologies are structured vocabularies having the possibility of reasoning. It includes definitions of basic concepts in the domain and relations among them. It is important that the definitions are machine-interpretable and can be processed by algorithms.

Why would someone want to develop an ontology?

Some of the reasons are [38]:

- To share common understanding of the structure of information among people or software agents.
- To enable reuse of domain knowledge.
- To make domain assumptions explicit.
- To separate domain knowledge from the operational knowledge.
- To analyze domain knowledge.

For an ontology being successfully used in requirements checking, it has to have the following properties: completeness, correctness, consistency, and unambiguity.

The intuitive meaning is:

- correctness means that the knowledge in the ontology does not violate the domain rules that correctly represent the reality;
- consistency means that there are no contradictory definitions in ontology;
- completeness means that the knowledge in ontology describes all aspects of the domain;
- unambiguity means that the ontology has defined a unique or unambiguous terminology.

There are not obscure definitions of ontology concepts, i.e. each entity is denoted by only one, unique name, all names are clearly defined and have the same meaning for the analyst and all stakeholders [38].

1.1.2 The concept of ontology

Many authors in their researches and articles may propose different ontology meaning and definitions. It is important to keep in mind that the definition of ontology mostly depends on the purpose why ontology is used and the task how it is used. In a case of this research, ontology meaning will be expressed from information systems and requirements engineering perspective.

Ontology from philosophy perspective, is the "branch of metaphysics that concerns itself with what exists" [Blackburn, 1996 p.269]. This perspective was first introduced in 1613. In the newer approach, it is described as the science of what is, of the kinds and structures of objects, properties, events, processes, and relations in every area of reality (Smith 2002). Philosophers have been studying ontology since Aristotle.

In the computer and information science, the term "ontology" occurred in a literature already in 1967, in the work on the foundations of data modelling by S. H. Mealy [74].

In recent year ontologies – shared conceptualizations of some domain – are increasingly seen as the key to further automation of information processing. There are many applications of such an approach, e.g. automated information processing, information integration or knowledge management, to name just a few. Especially after Tim Berners-Lee coined the vision of the Semantic Web, where Ewb pages are annotated by ontology-based meta-data, the interest in ontology research increased, in hope of finding ways to off-load large-volume information processing from the human user to autonomous agents [2].

The word ontology comes from the Greek ontos (being) and logos (word). It denotes the science of being and the descriptions for the organization, designation and categorization of existence [33]. Carried over to computer science in the field of artificial intelligence and information technologies, an ontology is understood as:

- a representational *artifact* for specifying the semantics;
- *meaning about the information or* knowledge *in a* certain domain *in a structured form [34] [37]*.

A theoretical model of ontology is adapted from the work of Kalibatiene (2009) and is presented in Fig. 1 [74].

A concept (by different authors also called a class) is an abstract or general idea inferred or derived from specific instances. A concept defines a set of individuals that belong together because they share the same properties (OWL2 2009). The most general concept named "Thing" is a concept of all individuals and is a super concept of all concepts [74].

Properties can be used to state relationships between individuals or from individuals to data values (OWL2 2009). Examples of properties include is-a, partof, subClassOf, equivalent, differentFrom, hasAge. Is-a, part of, subClassOf, equivalent, differentFrom relationships can be used to relate one concept to another, and the last one (has Age property) can be used to relate an instance of the concept to an instance of the data type. A set of properties depends on the chosen ontology language [74].

According to IDEF5 (2010), axioms are defined as precise characterization of the logic of a concept or a set of related concepts. An axiom typically expresses a constraint on the objects denoted by the terms in axiom.

The examples of axioms are reflexivity of relations, symmetry of relations, transitivity of relations, inverse relations, composition of relations, axioms implemented by special language (for example, PAL), etc. More detailed information about axioms can be found in the article of Vasilecas *et al.* (2009) [74].

In other literature resources, an ontology is stated as an explicit specification of a conceptualization [12]. It is a designed artifact that formally represents agreed semantics of a domain interest in computer resources [12]. This enables the sharing and reuse of information and allows for the interoperation of information systems [13].

But the most acceptable ontology definition is proposed in the specification prepared by OMG (2009) [74]:

An ontology defines common terms and concepts (meaning) used to describe and represent an area of knowledge. An ontology can range in expressivity from a taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a thesaurus (words and synonyms), to a conceptual model (with more complex knowledge), to a logical theory (with very rich, complex, consistent, and meaningful knowledge).

The structure of ontology can be defined mathematically. However, different authors provide different definitions. Ontology O is mostly defined as a triplet:

O = (C, P, A) ,

where C is a nonempty set of concepts, P is a set of properties and A is a set of axioms [74].

Although not a new field, ontology research has recently received renewed interest and attracted many other fields such as the semantic web, databases, electronic commerce, knowledge management, electronic learning, information retrieval, digital library, bioinformatics, geographical information systems, software engineering, intelligent systems and natural language processing. Thus, we can classify the ontology applications as reported in Pisanelli et al. [14], Fensel [15], Mizoguchi [1] and the most comprehensive survey by Hart et al. [16].

Mizoguchi [1] defines five typical types of ontology application including:

a) ontology as a common vocabulary;

b) ontology as assisting of information access;

c) ontology as the medium for mutual understanding;

d) ontology as specification;

e) ontology as foundation of knowledge systematization.

Van Heijst *et al.* (1996) classify ontologies according to their use [74]:

• Terminological ontologies, which specify what terms are used to represent the knowledge;

• Information ontologies, which specify storage structure data;

• Knowledge modelling ontologies, which specify the conceptualization of knowledge.

Fensel (2004) classifies ontologies:

• Domain ontologies, which capture the knowledge valid for a particular domain;

• Metadata ontologies, which provide a vocabulary for describing the content of online information sources;

• Generic or common sense ontologies, which capture general knowledge about the world providing basic notions and concepts for things like time, space, state, event, etc;

• Representational ontologies that define the basic concepts for the representation of knowledge;

• Method and particular tasks ontologies, which provide terms specific for particular tasks and methods. They provide a reasoning point of view on domain knowledge.

Ontologies, for software design and development, can be used with the following objectives [29] [26]:

• Specification: ontologies are used to specify either the requirements and components definitions (informal use) or the system´s functionality.

• Confidence: ontologies are used to check the system´s design.

• Reusability: ontologies could be organized in modules to define domains, subdomains and their related tasks, which could be later reused and/or adapted to other problems.

• Search: ontologies are used as information repositories.

• Reliability: ontologies could be used in (semi)–automatic consistency checking.

• Maintenance: ontologies improve documentation use and storage for system's maintenance.

• Knowledge acquisition: ontologies could be used as a guide for the knowledge acquisition process. Within Software Engineering, two main roles for ontologies have been considered [30]:

• Ontologies for the Software Engineering Process: the definition, re–use and integration of software components is aided by the use of ontologies as the conceptual basis.

• Ontologies for the Software Engineering Domain: the use of ontologies to describe the structure and terminology of the software engineering domain itself.

Ontologies can be classified according to the task they are meant to fulfill [42]:

- Knowledge representation ontologies describe the modeling primitives applicable for knowledge formalization;
- Top-level ontologies, also called upper-level ontologies, try to comprehensively capture knowledge about the world in general, describing for example: space, time, object, event or action, and so forth, independently of a particular domain;

- Domain ontologies and task ontologies contain reusable vocabularies with their relations describing a specific domain or activity. They can specialize the terms of top-level ontologies.

Ontologies, which are formal, explicit specifications of shared conceptualisations, encourage collaborative development by different experts. Ontologies capture knowledge at the conceptual level, thus enabling ID experts to directly manipulate them without the involvement of a knowledge engineer. In its simplest form, an ontology is a taxonomy of terms (i.e. a "shared lexicon") whereas more expressive approaches such as the Ontology Web Language OWL (W3C – OWL, 2003) encode knowledge in logical axioms. Ontologies support knowledge reuse by allowing more specific concepts to inherit the properties of those concepts they specialise. This also allows the representation of knowledge at different abstraction levels. In this way, instructional theories at a high level of abstraction can be related to concrete teaching methods [4].

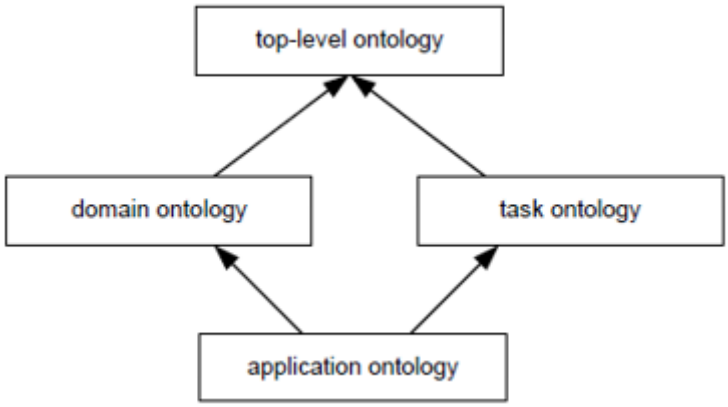
Ontology–driven software development or engineering has been defined as an approach that, based on ontologies, takes into account semantic constraints, adapting in a dynamic way to new constraints [28]. It could be considered a particular case of model–driven software, where models are based on ontologies at different levels of abstraction [26].

Based on these considerations, in (Guarino, 1998), the author proposes a classification of ontology kinds based on their level of dependence on a particular task or point of view (Guizzardi thesis):

Top-level ontologies describe very general concepts like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain;

Domain ontologies and *task ontologies* describe, respectively, the vocabulary related to a generic domain (like medicine, or automobiles) or a generic task or activity (like diagnosing or selling), by specializing the terms introduced in a top-level ontology;

Application ontologies describe concepts that depend both on a particular domain and task, and often combine specializations of *both* the corresponding domain and task ontologies. These concepts often correspond to *roles* played by domain entities while performing a certain task, like *replaceable unit* or *spare component*.



Reference: [75]

Fig. 3 Ontology types

Ontology may be classified as follows, based on the scope of the ontology (see also figure 3):

Upper/Top-level Ontology: it describes general knowledge (i.e. what time is and what space is) [75]:

- Domain Ontology: it describes the domain (medical domain, personal computer domain or electrical engineering domain).
- Task Ontology: it is suitable for a specific task (assembling parts together).
- Application Ontology: it is developed for a specific application (assembling personal computers).

Modularization can be used at each level. For instance, upper ontology could includes modules for Real Numbers, Time and Space (parts of Upper Ontology, generally are called generic Ontologies).Upper levels Ontologies could be imported by Ontologies at lower levels and adding them specific knowledge. Domain and Task Ontologies may be independent and are combined to be used for application ontology [75].

Benefits and problems of using ontologies in software engineering

Ontologies provide benefits regarding the process of software development, which could be summarized as follows, [31] [29] [26]:

• Ontologies provide a representation vocabulary specialized for the software process, eliminating conceptual and terminological mismatches.

• The use of ontologies and alignment techniques allow to solve compatibility problems without having to change existing models.

• Ontologies might help to develop benchmarks of software process by collecting data on the Internet and the use of the Semantic Web.

• Ontologies allow both to transfer knowledge and to simplify the development cycle from project to project.

• Ontologies promote common understanding among software developers, as well as being used as domain models.

• Ontologies allow for an easier knowledge acquisition process, by sharing a same conceptualization for different software applications.

• Ontologies allow to reduce terminological and conceptual mismatches, by forcing to share understanding and communications among different users during the ontological analysis.

• Ontologies also provide for a refined communication between tools forming part of an environment.

• Ontologies, when as machine–understandable representations, help in the development of tools for software engineering activities.

Although ontologies are considered a useful element within software engineering activities, some issues should still be born in mind when developing ontology–based software development projects [30] [26]:

• The ontology–based approach is adequate for those software development projects that belong to a set of projects within the same domain.

• The ontology–based approach allow to extend the notion of reusability to the modeling phase, not only the usual implementation one. Therefore, ontologies could be considered reusable model components in the system.

• Model–Driven Developments can benefit from the use of ontologies as model re–use mechanisms.

• The ontology–based approach affects all the software development process phases, from requirement analysis and domain analysis to integration, deployment and use of the developed software.

• The ontology–based approach allow ontologies to be used to facilitate software development in the long term, as well as addressing interoperability and re–use issues.

Furthermore, ontologies should exhibit some specific properties to facilitate their use within the software engineering community:

• Completeness: to assure that all areas of software development are covered. It could be achieved by paying attention to the different activities carried out by software development enterprises.

• Unambiguity: to avoid misinterpretations. Ambiguity could be avoided by using both concise definitions of concepts and semi–formal models.

• Intuitive: to specify concepts familiar to users' domain.

• Genericity: to allow the ontology to be used in different contexts. It could be done by keeping the ontology as small as possible, to achieve maximum expressiveness while being minimal.

• Extendability: to facilitate the addition of new concepts. It could be achieved by providing appropriate mechanisms defining how to extend the ontology.

The relevant problems identified by Mizoguchi and Bourdeau (2000) include [1]:

1. The "conceptual gap" between authoring systems and authors. In Artificial intelligence, the "knowledge level" is explicitly distinguished from the "symbol level" in which knowledge is encoded (Newell, 1982). The traditional approach to the development of rule-based systems involves a knowledge engineer, who encodes the knowledge elicited from a domain expert. Unfortunately, this results in a gap between the conceptualisation of the ID expert and the corresponding computer representation. Consequently, (i) the development, (ii) the verification & validation, and (iii) the maintenance of rule bases can become rather difficult.

2. The lack of theory awareness of systems. Heuristic rules cannot explicitly represent the theories they commit to.

3. The difficulty to integrate the latest research results. The lack of theory awareness prevents the adaptation of rule bases in order to accommodate subsequent results of ID research [4].

Summary

Well established requirements engineering process is one of the key factors for successful software development result. The result of the requirements engineering process is a requirements specification document. System requirements specification to be stated as "good", has to meet several formal criteria according to IEEE. And to meet

this criteria, overcome requirements engineering pitfals, widely known methodologies can be adjusted. In our case ontology will be established in the process to overcome the problems in requirements engineering.

Ontology is a field that can be found in philosophy, also in computer science, which is mostly interest for this research.

A lot of authors introduce definition of ontology and there is no common, general one exact description, but mostly agreed that it is shared conceptualizations of some domain. It is the field of artificial intelligence and information technologies, and it is understood as:

- a representational artifact for specifying the semantics;
- meaning about the information or knowledge in a certain domain in a structured form.

Ontologies have different purpose to be created, but mostly it is recognized as a knowledge base and a tool to formalize natural language to reuse it in computer technologies.

It can have different purposes, but also can have different layers of formality, such as top-level, domain and application ontology.

1.2 A comparative study on ontology languages and tools

To analyse ontology use in designing requirements specifications comprehensively, a comparative study on latest methodologies, languages and tools was held and described in this chapter.

1.2.1 Languages

For ontology representation in a machine-interpretable way, different languages exist. Ontology languages are usually declarative languages commonly based on either first-order logic or on description logic. Ontology languages based on first-order logic have high expressive power, but computational properties such as decidability are not always achieved due to the complexity of reasoning [35]. The most popular language based on description logic is OWL DL, which have attractive and well-understood computational properties [36]. Another relevant language in Ontological Engineering is the Resource Description Framework (RDF). RDF was originally meant to represent metadata about web resources, but it can also be used to link information stored in any information source with semantics defined in an ontology.

Many ontology languages have been developed, each aimed at solving particular aspects of conceptual modeling. Some of the, such as RDF(S) are simple languages offering elementary support for ontology modeling for the Semantic Web. There are other, more complex languages with roots in formal logic, focused around inference – ways to automatically infer facts not explicitly present in the model. Let's overview several languages created by W3 consortium:

• KIF: short for Knowledge Interchange Format, is a language based on first order logic created in 1992 as an interchange format for diverse knowledge related systems [75].

• RDF: stands for Resource Description Framework, was developed by the W3C to describe Web resources and allows the specification of the semantics of data based on XML in a Homogeneous, Interoperable Manner. It also provides mechanisms to clearly represent Services, Processes and Business Models, while allowing recognition of information not clear [75].

• RDFS: stands for RDF Schema and was built by the W3C as an extension to RDF with Frame-Based Primitives. RDF(S) is obtained by the combination of both RDF and RDF Schema. RDF(S) just allows the representation of Concepts,Taxonomies of Concepts and Binary Relations for that reason it is not very expressive. Three additional languages have been developed as extensions to RDF(S) and described in the following section (OIL, DAML OIL and OWL) [75].

• DAML+OIL: Stands for DARPA Agent Markup Language+. DAML+OIL, has been developed by a joint committee from the US and the European Union (IST) in the context of DAML, a DARPA project for allowing semantic interoperability in XML. Therefore, the same objectives as OIL are shared by DAML+OIL, it is built on RDF(S). DAML+OIL language was based on OIL as indicated by its name. The OIL and DAML+OIL languages allow Concepts, Taxonomies, Functions, Binary Relations and Instances representation. The tools that can author DAML+OIL Ontologies are OILEd, OntoEdit, Protégé2000 and WebODE [75].

• OWL: stands for Web Ontology Language, created in 2001 by a working group formed by W3C. The formed group has defined a list of main use cases for the Semantic Web, taking into account the DAML+OIL features as the main input for developing OWL and proposing the first specification of this language. Currently OWL

may be distinguished between OWL1 and OWL2, OWL1 includes three classes:OWL Full, OWL DL and OWL Lite [75].

• CycL: CycL was developed by Cycorp and it is a formal language whose syntax is a derivative from first-order predicate calculus and that extends first-order logic based on the second order concepts. CycL is adopted to express common sense knowledge and to represent the knowledge stored in the Knowledge Base Cyc. The CycL vocabulary includes terms: Semantic Constants, Integer, Strings, Non-Atomic Terms, Variables, etc. A knowledge base can be formed by a set of sentences. In brief, CycL uses predicate logic extended by typing, reification and microtheories that define a context for the truth of formulas [75].

To analyse ontology languages, formal criteria were presented by several authors [72], [73], [74] in their researches. These criteria are applicable to the research:

- **Specification perspective** – different languages may focus on different perspectives, and may provide constructs for only some perspectives. Authors of [72] define seven specification perspectives: a structural (a static structure); a functional (processes, activities and transformations); behavioural (states and transitions between them); a rule (rules for certain processes, activities, entities); an object (objects, processes and classes); a communication (language actions, meaning and agreement); and actor and role (actors, roles, societies, organizations). A structural perspective is the most important in ontologies.
- **Expressiveness** – possibility to express semantics (domain knowledge). I.e. according to the content of a domain knowledge, which can be expressed by a language, it can be lightweight, light heavyweight, and heavyweight. Another important aspect is lexical support – a capability for lexical referencing of elements (e.g., synonyms).
- **Inference engine** – possibility to inference new knowledge from the existing, i.e. possibility of reasoning.
- **Constraint checking** – existence of a constraint checking mechanism.
- **Mapping** with other languages. Nowadays, the feature of a language to map with other language is significant, because of reusability of knowledge and interoperability.
- **Standard** – describes if the language accepted as a standard.
- **Tools** – describes if a language implemented into a particular tool or not.

- **Formal syntax and semantics** - a formal language is made of three components: the syntax (e.g. rules for determining the grammatical well-formed sentences), the semantics (e.g. rules for interpreting sentences in a precise, meaningful way within the domain considered), and the pragmatic (e.g. rules for explaining how to use the language and for inferring useful information from the specification). Otherwise, a language is informal.
- **Popularity** – we add this criterion to determine the popularity of a language. We determine it according to the number of links in Google Scholar [73].

As can be seen, ten criteria are selected to compare ontology languages. However, it is important to note that some of those criteria are dependent from other but interdependence of criteria is not in the scope of this analysis.

The selection of applicable ontology languages was based on a brief literature study, choosing the criteria that best suits this research purposes.

In the table below, a comparative study on several ontology languages are presented. Languages were compared according to selected criteria, by their formality, available tools for developing ontology, expressiveness, specification perspective, inference engine availability, constraint checking opportunities, formal syntax and semantics, standard criteria for a language and popularity, which shows how often languages are used in the researches according to Google scholar. And also the novelty of the language, it should be not very old, as it is still supported and widely used these days, in a very changing technological environment. As mentioned abowe, it was chosen to compare OWL, RDF, CycL, DAML+OIL and KIF languages.

Table 3. Comparative study on ontology languages

Languages / **Criteria**	OWL	RDF	CycL	DAML+OIL	KIF
Standard	Yes	Yes	No	No	Yes
Tools	Many	Many	Few	Few	Few
Expressiveness	High	Medium	High	Medium	High
Specification perspective	Structural	Structural	Structural	Structural	Structural
Inference engine	Yes	No	Weak	Possible	Yes
Constraint checking	Good	Weak	Good	Weak	Weak
Mapping	RDF	OWL, DAML+OIL	OIL	RDF	No

Formal syntax	Yes	Yes	Yes	Yes	Yes
Formal semantics	Yes	Yes	Yes	Yes	Yes
Popularity	217 000	228 000	9 290	13 700	16 300
State of the art	Yes	Yes	Yes	Yes	No

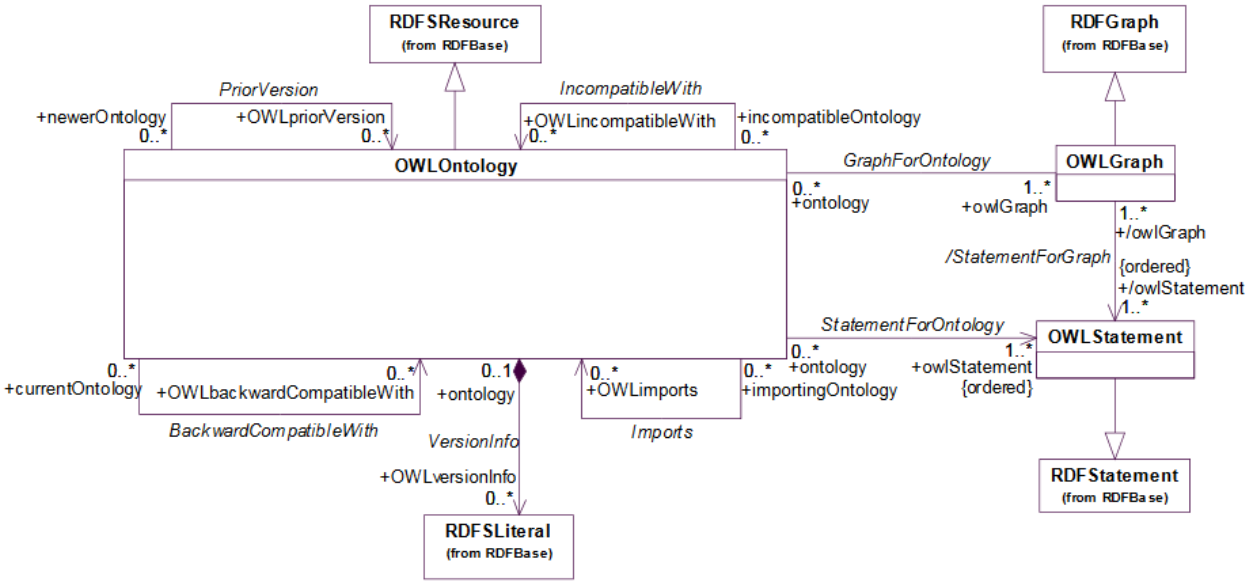
According to the study it was chosen to work with OWL, as it satisfies our criteria, is a standard, has tools, has a high expressive power, which is important for designing requirements, also it has structural specification perspective, as it is important to structure requirements and have reuse factor. It has mapping with RDF which will be additional language for the experiment, as they both are very related in some cases, such as data expression. It has formal syntax and semantics, it is quite popular and still supported as it is new age language. It also has DAML+OIL features, is like an improved version of the mentioned languages. And the last but very important criteria is that, it is created by W3 and is continuously improving also by OMG.

OWL

The W3C Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be exploited by computer programs, e.g., to verify the consistency of that knowledge or to make implicit knowledge explicit. OWL documents, known as ontologies, can be published in the World Wide Web and may refer to or be referred from other OWL ontologies. OWL is part of the W3C's Semantic Web technology stack, which includes RDF, RDFS, SPARQL, etc. (https://www.w3.org/OWL/).

OWL 2 is not a programming language: OWL 2 is declarative, i.e. it describes a state of affairs in a logical way. Appropriate tools (so-called reasoners) can then be used to infer further information about that state of affairs. How these inferences are realized algorithmically is not part of the OWL document but depends on the specific implementations. Still, the correct answer to any such question is predetermined by the formal semantics (which comes in two versions: the Direct Semantics [OWL 2 Direct Semantics] and the RDF-Based Semantics [OWL 2 RDF-Based Semantics]).

Also, it is important to note, that OWL is referring to RDF. OWL graph is an RDF graph (OWL 2 2009). Not all RDF graphs are valid OWL graphs, however. The OWLGraph class specifies the subset of RDF graphs that are valid OWL graphs [74].



Reference: W3.org

Fig. 4 OWL ontology

As shown in Figure above, an OWL ontology consists of a collection of facts, axioms, and annotations, defined in terms of RDF graphs and statements. The ontologyID (in the form of the URI reference it has by virtue of being a resource) allows us to make statements about a particular ontology – including annotations such as the relationship between a particular ontology and other ontologies, version information, and so forth (https://www.w3.org/TR/owl2-syntax/).

OWL 2 is a knowledge representation language, designed to formulate, exchange and reason with knowledge about a domain of interest. Some fundamental notions should first be explained to understand how knowledge is represented in OWL 2. These basic notions are:

- *Axioms: the basic statements that an OWL ontology expresses;*
- *Entities: elements used to refer to real-world objects;*
- *Expressions: combinations of entities to form complex descriptions from basic ones.*

While OWL 2 aims to capture knowledge, the kind of "knowledge" that can be represented by OWL does of course not reflect all aspects of human knowledge. OWL can be considered as a powerful general-purpose modeling language for certain parts of human knowledge. The results of the modeling processes are called ontologies – a terminology that also helps to avoid confusion since the term "model" is often used in a rather different sense in knowledge representation.

There are OWL tools – reasoners – that can automatically compute consequences. The way ontological axioms interact can be very subtle and difficult for people to understand. This is both a strength and a weakness of OWL 2. It is a strength because OWL 2 tools can discover information that a person would not have spotted. This allows knowledge engineers to model more directly and the system to provide useful feedback and critique of the modeling. It is a weakness because it is comparatively difficult for humans to immediately foresee the actual effect of various constructs in various combinations. Tool support ameliorates the situation but successful knowledge engineering often still requires some amount of training and experience (https://www.w3.org/TR/2012/REC-owl2-primer-20121211/#OWL_Syntaxes).

OWL provides three increasingly expressive sublanguages designed for use by specific communities of users and implementors:

- OWL Lite - which supports users primarily needing a classification hierarchy and simple constraints.
- OWL DL - which supports users who want maximum expressiveness without losing computational completeness and decidability of reasoning systems.
- OWL Full - which is intended for users who want maximum expressiveness and the syntactic freedom of RDF without computational guarantees.

1.1.3 Tools

To successfully develop an ontology, the tool is a must. In this chapter, several ontology tools will be represented and the most relevant to the research will be proposed, according to comparable study.

• **OntoEdit**: OntoEdit is an ontology editor integrating various aspects of ontology engineering. OntoEdit is quite exceptional in its category since it is based on a modern method for ontology development and because it makes comprehensive use of inferencing [75].

• **Protégé**: Protégé is an ontology editor created at Stanford University and is very popular in the field of Semantic Web and the level of computer science research. Protégé is free, developed in Java and its source code is released under a free license (the Mozilla Public License). Protégé can read and save ontologies in most ontologies

formats: RDF, RDFS, OWL, etc. It has several competitors such as Hozo11, OntoEdit and Swoop. It is recognized for its ability to work on large Ontologies [75].

• **OILEd**: OIL Editor (OilEd) is a simple ontology editor that supports OIL-based Ontologies construction. The basic design has been deeply influenced by similar tools such as Protégé5 and OntoEdit, but OilEd extended these approaches in several manners, especially using an extension of expressive power and a reasoner. OilEd supports the construction of OILbased Ontologies as an ontology editor [75].

• **Ontolingua**: The Ontolingua is an ontology tool created the Knowledge System Laboratory at Stanford University. Ontolingua is devoted for Ontologies development using a form-based Web interface. The ontology editor of Ontolingua is a tool supporting distributed, browsing,collaborative editing and Ontologies creation.Using Ontolingua, it is possible to export or import the following formats: KIF, DAML+OIL, OKBC, Prolog, LOOM, Ontolingua and CLIPS (C Language Integrated Production System). Additionally, it is also possible to only import Classic Ocelot and Protégé format, but not their export [75].

• **WebODE**: WebODE,can be defined as described in the Ontological Engineering Group webpage, “WebODE was built as a Scalable, Extensible, Integrated workbench that covers and gave support to most of the activities involved in the ontology development process (conceptualization, reasoning, exchange, etc.) and supplied a comprehensive set of ontology related services that permit interoperation with other information systems”. WebODE exports to WebODE’s XML, RDF(S), Prolog, OIL, Java/Jess, DAML+OIL, X-CARIN, UML and OWL, and imports from WebODE’s XML, RDF(S), UML, X-CARIN and OWL [75].

• **WebOnto**: WebOnto is a tool which provides a web-based visualization, browsing and editing support to develop and maintain Ontologies and knowledge models specified in OCML. An ontology can be viewed as a model of the conceptual structure of some domain and WebOnto provides the capability to represent this graphically [75].

• **OWLGrEd**: short for OWL Graphical Editor is a free UML style graphical editor for OWL Ontologies. It has further features for the exploration and development

of graphical ontology. OWLGrEd provides a complete graphical notation for OWL2, based on UML class diagrams and take into account the interoperability with Protégé [75].

• **Graffoo**: Graffoo stands for Graphical Framework for OWL Ontologies, is a superb new open source tool developed by Silvio Peroni that can be used to present the classes, properties and restrictions within OWL ontologies, or sub-sections of them, as clear and easy-to-understand diagrams. Several Graffoo diagrams have been developed to explain SPAR ontologies, or portions of them, and are to be found in the appropriate ontology directories [75].

To compare these described tools, several criteria was chosen, based on the study in [75]. First of all an important aspect is release date, because it shows how new and modern the tool is. The older it is, the more likely it won't be supported anymore. Second aspect is base language, on which language the tool is developed. If we choose one language, but the tool is based on another, it will be difficult to develop an ontology, it could come up to some problems like mapping. Also to what languages the tool can import and export data. It is also related to the base language and the tool facility. Availability is also an important aspect, because we do not want to experiment on niche tools, we would like to have a community for support and less likely it to be licenced as it would blow up the budget of the research. And the last ones, but not less important are ontology storage and ontology library aspects, as we would like the tool to be specifically developed for creating ontologies, to be specialized to work on ontology development problems, and the data we will later use on the further experiments, like files, code, etc. And it should have an ontology library already stored, as we would like to experiment with it for knowledge structure and reuse parts.

Table 4. Comparative study on ontology tools

Tool	Release date	Base language	Export/import to languages	Ontology storage	Availability	Ontology library
OILEd	31/10/2003	DAML+OIL	RDF URI's; limited XML Schema, export: HTML.	Files	Open source	Yes
OntoEdit	04/03/2004	F-Logic	RDFS, F-Logic, DAML+OIL; RDB, schemas	Files	Open source	No
Protégé2000	22/06/2004	OKBC+ CLOS	RDF, RDFS, DAML+OIL; XML, OWL, Clips; UML	Files &DBMs (JDBC)	Open source	Yes

		based metamodel				
WebODE	03/2002	HTML forms and Java applets	Imp/exp: XML, RDF(S), XCARIN, OWL Exp: OIL DAML + OIL FLogic, Prolog Jess, Java	DBMS (JDBC)	Open source	No
WebOnto	05/2001	OCML	Imp/exp: OCML Exp: Ontolingua GXL, RDF(S), OIL	Files	Open source	Yes
Ontolingua	11/2001	Ontolingua	Imp/Exp: KIF, OKBC,Loom,Prolog, Ontolingua, CLIPS import only: Ocelot, classic, Protégé	Files	Open source	Yes
OWLGrEd	10/ 2011	OWL	OWL, OWL2, UML, RDF/XML	Files	Open source	Yes
Grafoo	28/10/2013	OWL	OWL2, Turtle, RDF/XML, Manchester Syntax, or OWL/XML	Files	Open source	No

The best suitable tool for the research would be OWLGrEd, as it is released 6 years ago, is supported till these days and continuously improving. It is based on OWL language, which we chose in previous comparative study to work with. It perfectly exports and imports to languages as UML, RDF, XML, which as also very important to the research, as it will be used in research experiment. It has files storage, as it is very convenient for requirements specification development. It has ontology library, which is suitable for reuse existing knowledge and it is free to use. Also it as interperobility with Protégé, which will be used for ontology rules description.

Summary

To select the most suitable language and tool for the research case study, many articles analysis was held. And several criteria were selected, accorging to several authors. Different ontology languages and different tools serve different purpose and face different challenges.

The most relevant language for the research was chosen OWL 2, as it is a standard, it is an upgraded version of DAML+OIL, also it is quite modern and popular, it has formal syntax and semantics, well structured specification, has high expressive power, constraint checking availability. Also it is closely related and can be easily

mapped with RDF and XML, so that means it is provides broader range of opportunities.

With a tools, it was also important for tool to be modern, well supported till these days, to be based on OWL, as it will be used in the futher research. Should have import/export opportunities to RDF and UML. Should have file storage for specifications generation, should have a library of ontologies for existing knowledge reuse and also it is convenient that it is an open source tool, which means it is easily accessible and has community. At this point OWLGrED was chosen. Also it is important to note, that this tool has very expressive power for providing diagrams and quite convevient graphical interface. Also it as interperobility with Protégé, which will be used for ontology rules description.

1.3 The concept of domain metamodel

Requirements Engineering calls for an explicit domain knowledge. This domain knowledge generally resides in different areas, such as experiences, functionality, non-functional requirements, stakeholders and so on. Thus, it is necessary to concentrate this knowledge for the most appropriate application. Knowledge-driven techniques seem promising for this purpose. Kossmann et. al. in [66] define Knowledge-driven Requirements Engineering when Requirements Engineering is guided not only by a process but as well by knowledge about the process and the problem domain. In order to use knowledge-driven techniques, it is necessary to apply knowledge repositories that can be easily updated and utilised [39].

Furthermore, inferencing and decision support must be applicable on such a repository. Ontologies are one possible way for representing, organising and reasoning about the complex knowledge that requirements documents embody and have been proposed to be used in different ways for RE [39].

The benefits of the ontologies we already discussed in previous chapters. Now we will overview traditional architectures to integrate in the research.

1.3.1 Model Driven Architecture (MDA) to Ontologies

Model-driven Architecture (MDA) provides a framework for software development focusing on models in all phases of development [19]. Models are more than abstract descriptions of systems, as they are used for model- and code generation – they are the key part of the definition of a software system. Since in MDA abstract

models are refined to more concrete models, (automated) model transformations are very important [21]. For MDA methodologies we can distinguish two kinds of model transformations. In *vertical model transformations* models from higher level of abstraction are transformed to models of lower level of abstraction, e.g. platform independent models to platform specific models. There, knowledge of platforms is encoded into transformations, reused for many systems rather than redesigned for each new system. *Horizontal model transformations* are used for describing mappings between models of the same level of abstraction. By relating concepts of various model types, knowledge of modelling domains is encoded into transformations, enabling the integrated use of different models without having to specify interrelationships between each set models manually.

In MDA a model is a representation of a part of the functionality, structure and behaviour of a system. A specification is said to be *formal* when it is based on a language with well-defined structure ('syntax') and meaning ('semantics'). Thus MDA models must be paired unambiguously with a definition of the modeling language syntax and semantics [22]. Most metamodels have, despite of well-defined syntax, descriptions of their semantic concepts and dynamic semantics, which is neither formal nor machine understandable. Taking the idea of the semantic web [23], where the word semantic means machine understandable to modeling, metamodels have to be grounded using ontology meta data. This enables machines to understand the meaning of metamodels' concepts. In our approach we lift the syntactical (meta-)model description by semantic enrichment into ontologies describing the concepts of the model in a machine understandable form. Model transformations are defined on top of those ontologies [24].

By enriching model-driven development with ontologies a mutual understanding for conceptual integration can be achieved [25] [24].

Ontologies and MDA are two technologies being developed in parallel, but by different communities [5]. They have common points and issues and can be brought closer together [8] [5] [7]. Therefore, to bring software engineering practitioners and ontologies closer, many researchers suggest the use of Unified Modeling Languages (UML) in ontology development (e.g., [9] [10] [8] [7]). The main question they want to answer is how to use UML as well-accepted modeling languages for developing and using ontologies in real world applications. Although the ontology concepts are

coincidently similar to object-oriented paradigms, it has some limitations mainly regarding the concept of property. Because of these discrepancies, initially, we could only use UML in the beginning of ontology development. However, there is a significant movement in this research to overcome these limitations using UML extensions (i.e. UML profiles) as implemented in [8] [7]. As a result, the Object Management Group (OMG) has established Ontology Definition Metamodel (ODM) as a MDA standard metamodel for modeling ontology [8]. The ODM defines concrete abstract syntaxes (i.e. OWLDataTypeProperty, OWLClass) for modeling ontology that can be represented by using UML profiles [8]. The ODM is centrally based on UML and the W3C Web Ontology Language (OWL) recommendation [11]. In terms of ontology modeling, on one hand, the UML provides powerful graphical modeling capabilities and widely supported tools (i.e. Rational Rose, Poseidon, Magic Draw, ArgoUML, etc). In addition, since the UML and ODM are MOF-compliant languages, it is possible to store ontologies in MOF-based repositories, to store ontologies diagrams in a standard way (UML2 XMI), as well as to share and interchange ontologies using XMI [8] [7]. However, on the other hand, we note that not all OWL features could be represented by UML. We will use ODM and UML profiles defined in [8] for representing ontologies and designing the server. In addition, UML is currently a de facto standard modeling language. There is a growing interest in its adoption as a language for conceptual modeling and ontological representation (e.g., [9] [10] [8][7]).

The benefits of MDA are significant-to business leaders and developers alike (OMG 2009):

- Reduced cost throughout the application life-cycle;
- Reduced development time for new applications;
- Improved application quality;
- Increased return on technology investments;
- Rapid inclusion of emerging technology benefits into their existing systems.

MDA provides a solid framework that frees system infrastructures to evolve in response to a never-ending parade of platforms, while preserving and leveraging existing technology investments. It enables system integration strategies that are better, faster and cheaper (OMG 2009).

1.3.2 Ontology Development Metamodel (ODM)

A trigger for the call for development of an ODM was the development by the World-Wide Web Consortium of the Web Ontology Language OWL. OWL has a number of features which emphasize weaknesses in UML for ontology development, including:

- The ability to fully specify individuals apart from classes, and for individuals to have properties independently of any class they might be an instance of.
- The OWL property is much more flexible than the UML association. In particular it can be used to model complex mereotopological relationships and hence complex objects. (Mereotopological relationships are whole-part relationships, including those involving spatial parts and their geometric and topological relationships.)
- OWL Full allows classes to have instances which are themselves classes. (šaltinis: OMG ODM)

Furthermore, organizations developing ontologies will often build on legacy data models represented in UML or one of the dialects of Entity– Relationship (ER) Modeling, even if the development is carried on in one of the newer metamodels.

Since there are so many metamodels which a developer might need to take into account in an ontology project, the ODM Group decided that it would not be sufficient to develop a metamodel for OWL only, but instead to develop a suite of MOF metamodels, for RDFS/OWL, Topic Maps and CL. UML of course already has a MOF metamodel.

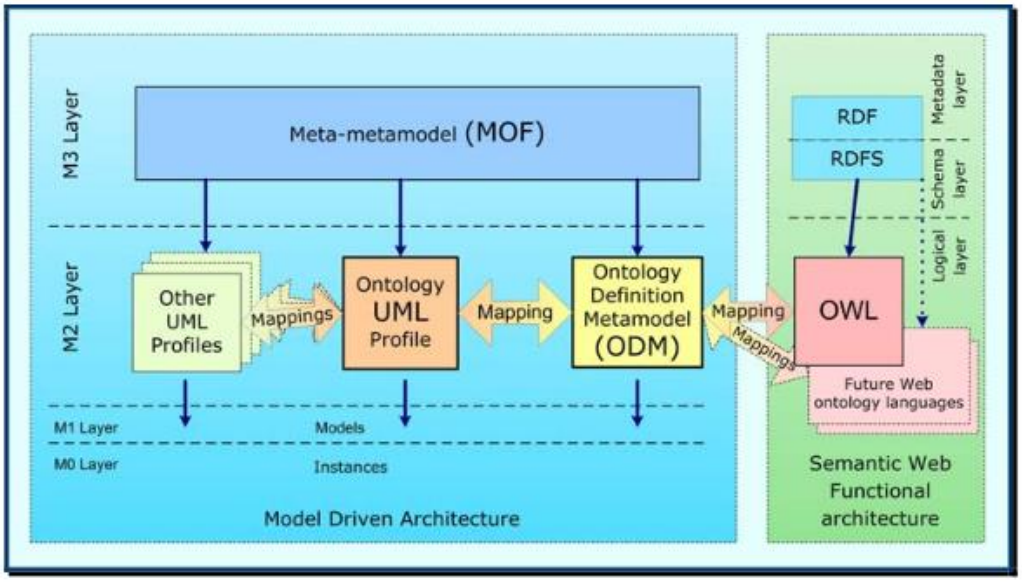


Fig. 5 Ontology in the context of MDA

The different metamodels express a concept quite differently. To show this difference, we will use a simple running example, illustrated in Fig. 8.2 as a UML model, of a simple model which might be a fragment of a university teaching ontology, namely that students enroll in courses (šaltinis: OMG ODM).

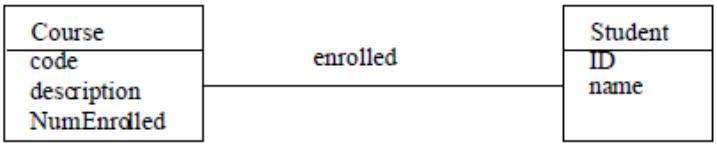


Fig. 6 Fragment of a university teaching ontology, expressed in UML

One of the advantages of UML, and hence the MOF, is that there is a well-established relationship between UML Class Diagrams and database schemas, implemented by many more or less automatic tools. This relationship allows a first cut at a repository for any of the metamodels in the ODM.

1.3.3 Enterprise metamodel

OMG provides Model Driven Architecture (MDA) approach to information systems engineering where MDA focuses on functional requirements and system architecture not on technical details only. Model Driven Architecture allows long-term flexibility of implementation, integration, maintenance, testing and simulation. It means that enterprise modeling and user requirements engineering stages of information system engineering life cycle are not covered enough by MDA yet. There is lack of formalized problem domain knowledge management and user requirements acquisition techniques for composition and verification of MDA models. In order to solve this problem enhancement of MDA approach by the best practices of Knowledge Base IS engineering (including Enterprise Knowledge repository) can be used. The proposed enhancement will intellectualize MDA models composition process by improving their consistency and decreasing the influence of the empirical information in composition process. Knowledge Base Subsystem ensures MDA models verification against formal criteria defined by Control Theory. It is believed to reduce risk of project failures caused by inconsistent user requirements and insufficient problem domain knowledge verification.

The researchers and scientists of Vilnius University developed a framework of Knowledge-based Enterprise model, which helps to generate models, that could be used for requirements specification. Knowledge-based CASE systems holding substantial components, which organize knowledge: knowledge-based subsystem's knowledge base, which essential elements are enterprise meta–model specification and Enterprise Model for certain problem domain. Enterprise Model as organization's knowledge repository enables generate UML models with the help of transformation algorithms. Enterprise meta-model holds essential elements of business modelling methodologies and techniques, which ensures a proper UML models generation process. In order to decrease the influence of empirical factors on IS development process, the decision was made to use knowledge-based IS engineering approach. The main advantage of this approach is the possibility to validate specified data stored in EM against formal criteria, in that way decreasing the possible issues and ensuring more effective IS development process compared to classical IS development methods. It could be stated that this metamodel is part of MDA approach, this is why it is relevant to this research and it will be used in our framework.

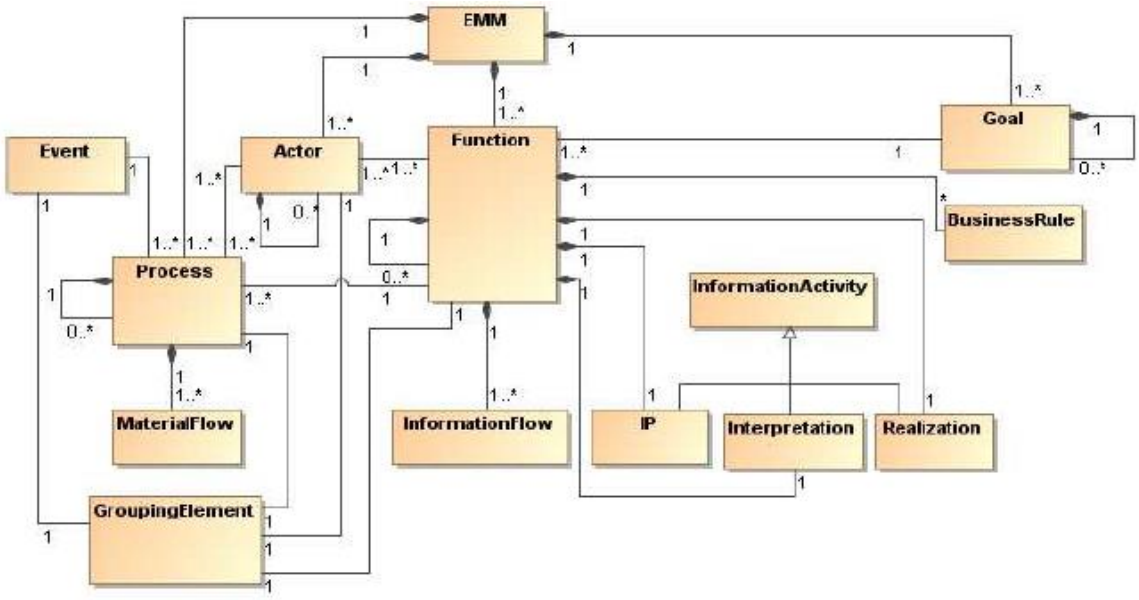


Fig. 7 Basic elements of Enterprise Meta-model

Knowledge Based Subsystem, which improves traditional MDA conception with best practices of problem domain knowledge and user requirements acquisition methods, is presented in Fig above. It ensures the problem domain knowledge verification against EMM internal structure. Such usage of Knowledge Based

Subsystem together with MDA improves the consistency of software artifacts and reduces IT projects dependency on empirical processes.

The EMM is intended to be formal structure and set of business rules aimed to integrate the domain knowledge for the IS engineering needs. It is used as the "normalized" knowledge architecture to control the process of construction of an EM.

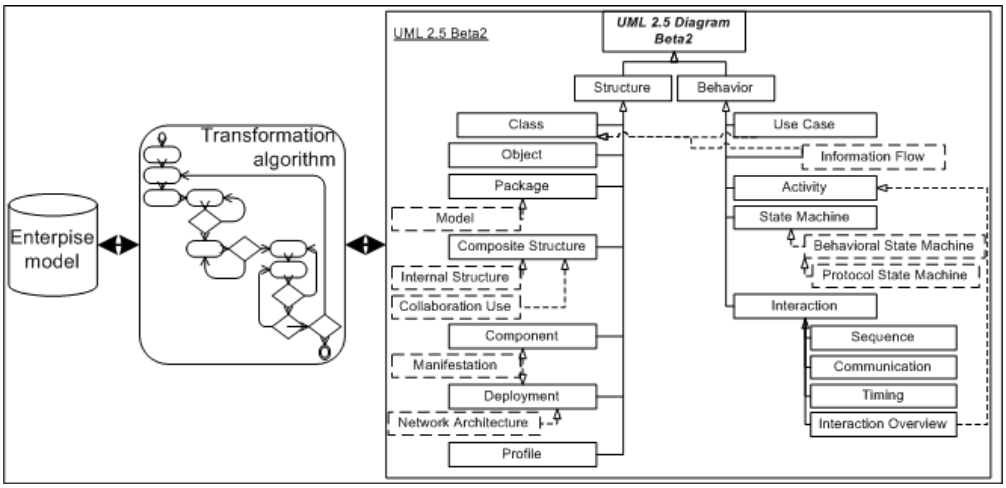


Fig. 8 UML models generation by using the transformation algorithm

EMM mostly focuses on consistency of UML models generation. Also it is used as knowledge repository, where domain knowledge is stored. It's structure can be easily adapted to any domain, which means it is easily reusable. That is a huge advantage for the research. But even though, it has advantages, it has some drawbacks in a scope of requirements engineering too:

- *It does not provide semantic concept of the requirements;*
- *It does not provide rules and logic for associations above requirements;*
- *It does not provide a shared common understanding of the structure of information among people or software agents;*
- *It does not provide rules for completeness, unambiguity and traceability criteria.*

For the problems mentioned above solving, EMM and ontology integration should be used. An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements.

Using ontologies with Enterprise Modelling offers several advantages. Ontologies ensure clarity, consistency, and structure to a model. They promote efficient

model definition and analysis. Generic enterprise ontologies allow for reusability and automation of components. A common ontology allows to ensure shared understanding, clearer communication, and more effective coordination among the various divisions of an enterprise. These lead to more efficient production and flexibility within the enterprise [76].

Summary

To structure domain knowledge, which is the key factor for successfully developed requirements specification, the methodology is needed. Requirements Engineering calls for an explicit domain knowledge. This domain knowledge generally resides in different areas, such as experiences, functionality, non-functional requirements, stakeholders and so on. Thus, it is necessary to concentrate this knowledge for the most appropriate application. Knowledge-driven techniques seem promising for this purpose. Kossmann et. al. in [66] define Knowledge-driven Requirements Engineering when Requirements Engineering is guided not only by a process but as well by knowledge about the process and the problem domain. In order to use knowledge-driven techniques, it is necessary to apply knowledge repositories that can be easily updated and utilised [39].

Furthermore, inferencing and decision support must be applicable on such a repository. Ontologies are one possible way for representing, organising and reasoning about the complex knowledge that requirements documents embody and have been proposed to be used in different ways for RE [39].

MDA based EMM mostly focuses on consistency of UML models generation. Also it is used as knowledge repository, where domain knowledge is stored. It's structure can be easily adapted to any domain, which means it is easily reusable. That is a huge advantage for the research. But even though, it has advantages, it has some drawbacks in a scope of requirements engineering too:

- It does not provide semantic concept of the requirements;
- It does not provide rules and logic for associations above requirements;
- It does not provide a shared common understanding of the structure of information among people or software agents;
- It does not provide rules for completeness, unambiguity and traceability criteria.

For the problems mentioned above solving, EMM and ontology integration should be used. An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements.

Using ontologies with Enterprise Modelling offers several advantages. Ontologies ensure clarity, consistency, and structure to a model. They promote efficient model definition and analysis.

1.4 The concept of ontology based model transformations

In this chapter, deeper overview of the transformations will be proposed, as models transformations will be used in between OWL and UML, OWL and RDF, Enterprise metamodel and Requirements ontology. To establish a successfull method, transformations will be required in several points of methodology.

As a realization of semantic-based model transformations, ontology-based model transformation needs the following parts to achieve an increased level of abstraction [24]:

• Semantic Transformation: A *semantic transformation* is a transformation specification describing a transformation between two ontologies. A semantic transformation is specified between a source ontology and a target ontology (see figure 1), but it can also be bidirectional. For horizontal transformations the semantic transformation normally is the ID.

• Syntax-semantic Binding: The *syntax-semantic binding* specifies the connection between syntax (metamodels) and semantics (ontologies).

• MO-Binding: *(Metamodel-ontology) MO-Bindings* specify how semantic information can be derived model elements.

• OM-Binding: *(Ontology-metamodel) OM-Bindings* specify how ontology elements are expressed in models.

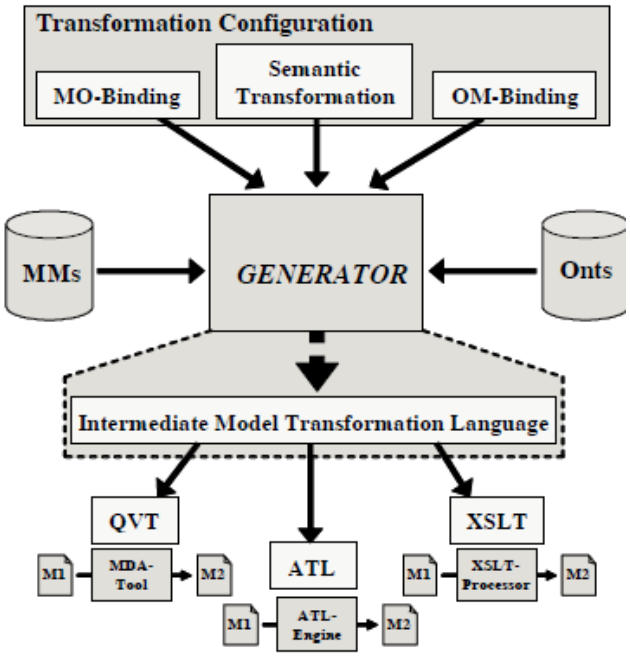


Fig. 9 Overall approach of ontology-based model transformation

In figure 9 we can see concepts and design of ontology-based model transformation. A transformation is specified on the basis of ontologies, called *semantic transformation*. The transformation between the two ontologies, a *source ontology* and a *target ontology*, is described by the means of this *semantic transformation*. Elements of the source ontology are transformed to elements of the target ontology. The connection between syntax defined in metamodels and the semantics of the ontology elements has to be defined by a syntax-semantic binding, done with a *MO-Binding* and an *OMBinding*. In a mid-term perspective these bindings have to be derived semiautomatically from already existing transformations and bindings in combination with metamodel analysis [24].

Figure 9 shows the overall approach of ontology-based model transformation. A combination of one semantic transformation, one MO-Binding and one OM-Binding form a *transformation configuration*. A transformation configuration is the basis for an automated generation of common model transformations. A generator for model transformations takes a transformation configuration as well as appropriate metamodel- and ontology-definitions as input and outputs a model transformation specified in an *intermediate model transformation language*. Introducing an intermediate transformation language aims to obtain a common representation of model transformations independent to specific transformation languages, maybe on the basis

of a QVT common language and comparable to the platform independent model in the MDA approach. The generated model transformation is input to arbitrary MDA-tools performing model transformations [24].

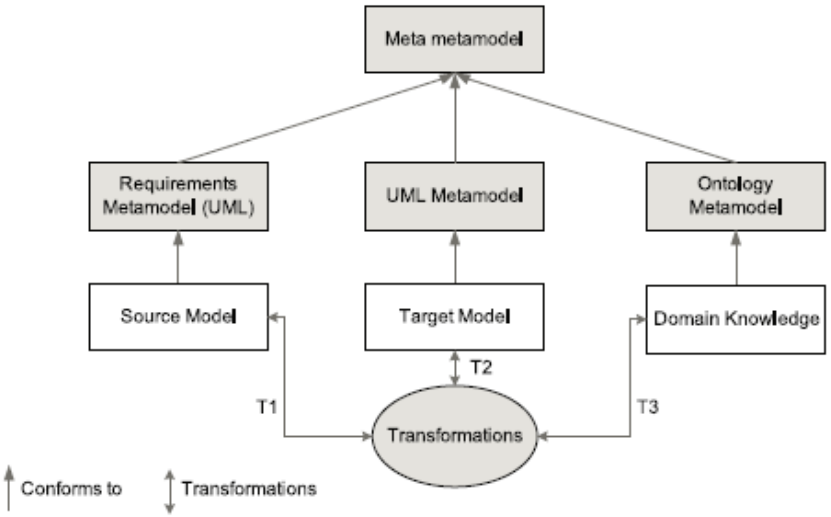


Fig. 10 Metamodel-based transformation

Model transformations specified between ontologies, will lead to interoperable model transformations independent of methodologies' tailoring to specific projects. The specification of multiple model transformations will be reduced to few or even one ontology-based model transformation. Furthermore, one specification of an ontology-based model transformation can be used to generate multiple transformations for specific modeling environments (and their transformation languages) automatically [24].

1.1.4 UML transformation to Ontology

An ontology is a kind of data model. The UML Class Diagram is a rich representation system, widely used, and well supported with software tools. Why not use UML for representing ontologies?

One reason is that a UML Class Diagram is a specification for a system. It shows schemas, but does not necessarily fully specify instances. Even if instances are fully specified, it is not common to represent a large population of concrete instances. We know that the shared worlds modeled with ontologies contain instances as well as schemas, for example the periodic table of the elements includes classes like rare earths and noble gases, but also individuals like hydrogen and helium. UML is intended to be used with some sort of implementation, like an SQL database manager, which

completes the specification of the instances, and represents and stores the concrete populations.

Further, a UML Class Diagram is generally used by the software engineers building a system as part of the design specification. It can be a component of a computer-aided software engineering tool which can automatically generate implementations. But class diagrams are not intended for public use, to be combined as components in larger ontologies, or to be used at run-time. It is of course possible to adapt UML to these purposes, but they are not part of its design.

Finally, and perhaps most importantly, an ontology by definition is intended to be reused, or to have multiple implementations across applications. While reuse is also an important aspect of the OMG's Model-Driven Architecture methodology, in the case of an ontology, the ability to unambiguously interpret the definitions and axioms expressed is essential to enabling automated reasoning. There must be some way of verifying that two implementations committed to a single ontology are logically consistent with one another. Common Logic and OWL enable this by having a formal semantics expressed as a model theory. Two implementations which generate the same objects by definition agree. UML does not at present have a published model theory or proof theory that would enable such automated validation or reasoning processes.

So this is why the OMG called for development of an ontology development metamodel distinct from UML.

1.1.5 Mappings

For example, there are two different ways to map *N*-ary associations from UML to OWL, depending on whether we take OWL Full or OWL DL as target. OWL has a mandatory universal superclass (owl:Thing) which can map to a universal superclass in UML, but this is contrary to normal practice in UML modeling. A particular project might analyze the uses of universal properties in the OWL source model and choose to declare a number of more general but not universal superclasses in the UML target.

In the W3C Semantic Web Best Practices working report on Topic Map mappings [14], the point is made several times that there are different ways to map particular structures, and that each way has its advantages and disadvantages. In any particular project, design decisions will be taken in favor of advantages and against disadvantages so different projects will map in different ways (šaltinis: OMG ODM).

The mapping strategy in the ODM is illustrated in Fig. 8.10. Note that there will be mappings from each metamodel to and from OWL Full, except for CL for which there is only a mapping from OWL Full.

Profiles and mappings are related. Consider these cases:

We use a MOF tool to develop an OWL ontology, which is then serialized using the XML markup XMI defined for the MOF. In this case we use the ODM OWL MOF model alone, and do not need mapping or profile.

We have a native UML model which we want to serialize as OWL XMI (using OWL-derived markups). In this case we use both the MOF UML and MOF OWL metamodels, together with the UML -> OWL mapping, but no profile.

We have an OWL-profiled UML model to be serialized as OWL XMI. Here we use the ODM OWL MOF model and the UML2 MOF model with the UML2 -> OWL mapping and information from the ODM OWL profile for UML.

These three are all useful scenarios. The third would be a more complete OWL model using UML notation than the second, while the first does not care about UML at all.

Further, if profiles are being used the modeler might want to use UML notation to create and visualize an ontology (say in OWL). This implies that two MOF models are required, one for UML and the other for OWL. The mapping UML -> OWL is required, because without application of a mapping the final result would be UML XMI rather than OWL XMI (šaltinis: OMG ODM).

The ontologies are used throughout the enterprise system development life cycle process to augment and enhance the target system as well as to support validation and maintenance. Such ontologies should be complementary to and augment other UML modeling artifacts developed as part of the enterprise software development process. Knowledge engineering requirements may include some ontology development for traditional domain, process, or service ontologies, but may also include:

- Generation of standard ontology descriptions (e.g., OWL) from UML models.
- Generation of UML models from standard ontology descriptions (e.g., OWL).
- Integration of standard ontology descriptions (e.g., OWL) with UML models.

Summary

To ensure successful MDA and ODM integration to our solution, model transformations are essential. This is quite demanding approach, because it has to be taken into account, that MDA relies on UML language and ODM relies on OWL language. But these two languages can be easily mapped together, there are many transformation engines to support that process.

1.5 Existing technologies in the research field

To develop a successfull method, existing methodologies should be reviewed, as they re presented for the same or similar problematics. Also, knowing the market of the research problematics, is a huge advantage.

Ontologies are rapidly growing technology in Requirement Engineering as well as in general. In the scope of the research, only ontologies that applies to requirements engineering will be analysed. The table of RE specific ontologies and methodologies are presented below.

Table 5. Ontologies in Software Engineering

Author(s)	Presented methodology	Description	Key features
Kossmann [66]	**OntoREM**	a comprehensive specification of the ODRE methodology, including the underlying concepts in the RE domain and relationships between them; consists of the OntoREM Metamodel ontology and a number of domain ontologies; All requirements are managed with DOORS2 from IBM; comes with a workflow for the RE process; concepts *OntoREMGoalHierarchy* and *OntoREMRequirement* with their described relationships define "templates" of goals, soft goals and requirements that are used when creating an instance of a goal, soft goal or requirement and are linked to the relevant areas of available domain ontologies.	Based on MDA; Cover domain knowledge; Create templates; Related to goals;
Jureta [58]	**CORE**	Capture basic stakeholder concerns during RE, namely beliefs, desires, intentions, and evaluations; the ontology grounds on the foundation ontology DOLCE; it proposes four relationships to relate instances of concepts in CORE: refine, approximate, compare and evaluate.	Not based on MDA; Does not cover domain knowledge; Covers only basic requirements.
Kayia [59]	**FRS Ontology**	method that allows for requirements analysis of a functional requirements specification; the method is based on a domain ontology and a mapping to the requirement specification; the ontology consists of a thesaurus and inference rules.	Not based on MDA; Cover domain knowledge; Cover only functional requirements.

Riechert [60]	**SWORE**	support the RE process semantically; provides a semantic structure for capturing requirements information and linking this information to domain- and application-specific vocabulary; in order to allow the various stakeholder a collaborative elicitation of requirements, SWORE has been integrated into the semantic collaboration platform SoftWiki. Unfortunatelly, this project is closed.	Not based on MDA; Cover domain knowledge; Formal semantics.
Kassab and Daneva in [65]	**NFR Ontology**	NFR ontology considers non-functional requirements early in software development. The NFR ontology defines the meaning of a set of concepts for the NFR domain. The ontology allows for capturing relationships of NFRs with functional requirements in the form of association points. NFRs can be further decomposed (AND/OR decomposition) and have operationalizations, that is a refinement of a NFR into a solution in the target system (operations, functions, data representations and architecture design decisions) that will satisfy the NFR.	Oriented to non-functional requirements.
Kof [62]	**NL approach**	A method to build a domain ontology from requirement documents provided in natural language. Therefore, terms are extracted from text and clustered, a taxonomy is built. Associations between the extracted terms are identified and make up together with the associated terms the domain model. While the formatting, tagging, parsing and concept cluster building are automatic, the identification of cluster intersection and taxonomy building as well as deciding which associations are sensible remains interactive and, thus, need human interaction.	Not based on MDA; Cover domain knowledge;
Ying et al. [63]	**Inconsistency checking algorithm**	An algorithm for detecting and resolving inconsistencies of domain ontologies for RE. The domain ontology is considered to be a thesaurus containing all the information about domain concepts and their role. Thus, inconsistency of domain knowledge can be found by ontology consistency checking. The algorithm is based on the Tableaux algorithm, consistency rules are formally defined and semantic checking is proposed to resolve detected inconsistencies. However, consistency checking is only performed regarding the logical consistency of the ontology. That is checking whether the ontology is satisfiable, which means that there is no contradicting information in the ontology.	Cover domain knowledge

Zhu et al. in [64]	**Requirements refinement tree**	An ontology-based approach for inconsistency measurement of requirements specifications based on a requirements refinement tree. Therefore, requirements are stepwise decomposed until a requirement can be realized. During this process of requirements refinement, external requirements from the customers are extracted first.	Not based on MDA. Does not cover domain knowledge
Eric Yu [50]	**i*/Tropos**	The agent-oriented modelling framework i* was developed for modelling and reasoning about organisational environments and their information systems. The framework can be used for several purposes, e.g. Requirements Engineering and Software Process Modelling.	Based on MDA; Does not cover domain knowledge; Goal oriented.
Darimont et al. [53]	**KAOS**	The KAOS (Knowledge Acquisition in automated Specification or Keep All Objects Satisfied methodology is a GORE approach. KAOS is described in as a multi-paradigm framework that allows to combine three levels of expression and reasoning: semi formal for modelling and structuring goals, qualitative for selections among alternatives and formal for more accurate reasoning when needed. A generic ontology forms a metamodel for requirements.	Not based on MDA; Does not cover domain knowledge; Goal oriented.
Farefelder [52]	**Ontology-driven guidance for requirements elicitation**	A prototype of a semantic guidance system that assists the requirements engineer in capturing requirements by using semi-formal representation. Their approach aims to prevent specifying and finally resolving incorrect requirements. Instead, the prototype automatically proposes at least parts of the requirements by using information originating from a domain ontology. On these suggestions the requirements engineer can build on to define requirements.	Cover domain knowledge
Castaneda et al. [40]	**OntoSRS**	A framework is divided into three application areas, such as: the description of requirements specification documents, the formal representation of the application domain knowledge, and the formal representation of requirements. Framework addresses the issues in RE, creates key points where to integrate ontologies in RE process, but more effort is needed to implement the framework.	Based on MDA; Cover domain knowledge; IEEE criteria oriented

Several ontology-based requiremnts engineering concepts were analyzed in this chapter. Key formal criteria were proposed, according to IEEE organization, standard 830. It is stated that these criteria are non-negotiable while speaking about good requirements specification. It could be extended, mapped, but the base is non-negotiable.

Mapping with already existing solutions and criteria was made during this analysis and results came up to the table below.

Table 6. Our solution and existing solutions compare

Criteria/ Methods	Correctness	Consistency	Unambiguity	Completeness	Extendability	Modifiable	Traceable
CORE	-	-	+	-	+	+	-
ontoREM (ODRE)	-	+	-	+	+	+	-
i*/Tropos	-	+	-	-	-	-	-
FRS	-	+	-	+	-	-	+
SWORE	+	-	-	+	+	+	-
NL approach	-	-	-	-	+	+	-
Req.refinement tree	+	-	-	+	-	-	-
NFR (non-func.)	-	-	+	-	-	-	-
Inconsistency checking rules	-	+	-	-	-	+	-
KAOS	-	-	-	-	+	+	-
OntoSRS	-	+	+	+	-	-	-
Guidance for req elicitation	+	+	-	-	-	-	-
Our solution	+	+	+	+	+/-	+/-	+

Several problems were concluded during this analysis, that already proposed methods do not cover:

- Requirement knowledge is not sufficiently covered. Intentions, risks, obstacles and decisions are not documented during RE and thus, are not available at later stages during software development.
- Most of the solutions do not meet correctness and traceability requirements. Also very few cover unambiguity and completeness criteria.
- Requirement problems (e.g. conflicts, unstated information) are detected too late or not all.
- To dig deeper into realisation of the methods, relationships among requirements are inadequately captured and are often limited to binary relations between requirements instead of defining which kind of relation is meant (e.g. excluding, alternative, generalization).
- Methods need richer and higher-level abstractions.
- Some of the methods are goal-oriented on requirements engineering, so that means it does not cover domain knowledge, scenario-based requirements.

- Some of the methods are incomplete or oriented only to functional or non-functional requirements.
- Not all of the methods and tools are still supported, which shows that they were not very successful and beneficial.
- Just a few methods are oriented to requirements analysis in the RE process. Mostly of them are oriented to requirements elicitation.

Also, one of the main problems which is very relevant to our research is that not all of the methods listed above are based on MDA architecture. Only ODRE implicates or mentions ISO standards base of the requirements engineering while developing the method.. Only OntoSRS gives brief reasoning of the method based on IEEE 830 standard.

According to these problems we propose a solution, that will cover them and improve the requirements specification quality. The most promising methods are ODRE and OntoSRS. OntoSRS is no longer supported, but several ideas will be taken into account while modelling our solution.

Our solution benefits:

- Domain knowledge oriented;
- Reusable and extendable;
- Will cover requirements analysis and specification phases;
- Structure the knowledge by using knowledge-based methods;
- Upgraded successful ODRE and OntoSRS methods;
- Will cover corectness, unambiguity, completeness and traceability by IEEE 830 criteria;
- Will structure the requirements specification according to well known IEEE 830 template to ensure correctness, unambiguity and completeness of the requirements;
- Ontology and Enterprise metamodel will be used as knowledge repository about domain;
- The solution will be based on the standard methodologies, such as MDA.

Summary

Requirements engineering process can have big benefits by adapting ontologies to it's technology. While analyzing requirements engineering processes, standards for correct SRS and already existing tools to support RE process, we came up to the conclusions, that many tools are already created and they are created to solve different problems, different aspects of RE. Some of them are based on the standards, some of them not.

Key argument why additional solution is needed is that in existing ones requirement knowledge is not sufficiently covered. Intentions, risks, obstacles and decisions are not documented during RE and thus, are not available at later stages during software development. This knowledge is covered by MDA use as a framework for a solution. Most of them covers only one small part of the RE, but as it is stated in IEEE 830 standard, which is applicable to ISO and SEBoK, for requirements specification to be consistent, complete and unambiguous, all of the related parts of it should be described and analysed. That is why the solution should be created from a broader vision to requirements engineering process and requiremens specification document.

1.6 Conclusions

Conclusions of the first chapter on literature review and analysis. In the first chapter, based on literature review and analysis, several definitions and approaches were introduced. Ontology meaning and concept was introduced, comparative analysis based on criteria of ontology languages and tools was held. Also requirements engineering concept and ontology integration into it was analysed. Existing methodologies were presented that are trying to solve variuous problems, related to requirements engineering concept. Domain metamodel importance was introduced and classic approaches described, as well as mapping and transformation rules.

An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements so difficult.

Requirements engineering process can have big benefits by adapting ontologies to it's technology. While analyzing requirements engineering processes, standards for correct SRS and already existing tools to support RE process, we came up to the conclusions, that many tools are already created and they are created to solve different

problems, different aspects of RE. Some of them are based on the standards, some of them not.

Key argument why additional solution is needed is that in existing ones requirement knowledge is not sufficiently covered. Intentions, risks, obstacles and decisions are not documented during RE and thus, are not available at later stages during software development. This knowledge is covered by MDA use as a framework for a solution. Most of them covers only one small part of the RE, but as it is stated in IEEE 830 standard, which is applicable to ISO and SEBoK, for requirements specification to be consistent, complete and unambiguous, all of the related parts of it should be described and analysed. That is why the solution should be created from a broader vision to requirements engineering process and requiremens specification document.

Requirements Engineering calls for an explicit domain knowledge. This domain knowledge generally resides in different areas, such as experiences, functionality, non-functional requirements, stakeholders and so on. Thus, it is necessary to concentrate this knowledge for the most appropriate application. Knowledge-driven techniques seem promising for this purpose. Knowledge-driven Requirements Engineering when Requirements Engineering is guided not only by a process but as well by knowledge about the process and the problem domain. In order to use knowledge-driven techniques, it is necessary to apply knowledge repositories that can be easily updated and utilised.

For domain knowledge repository, MDA based EMM was chosen as the most relevant approach as it stands out for classic methodologies. So combined MDA and ODM methodologies, we can get great results. An ontology-based requirements specification tool may help to reduce misunderstanding, missed information, and help to overcome some of the barriers that make successful acquisition of requirements. Using ontologies with Enterprise Modelling offers several advantages. Ontologies ensure clarity, consistency, and structure to a model. They promote efficient model definition and analysis.

These conclusions leads to the methodology of the research to be presented in the second chapter.

2 Other activities during 2015-2017 year of study

Exams taken

Title of the lecture	Credits	Planned date	Exact date	Grade
Informatikos ir informatikos inžinerijos tyrimo metodai ir metodika	9	2016.05.20.	2016.06.09	6
Informacijos poreikių specifikavimas	7	2017.03.	2016.09.27	9
Žiniomis grindžiama kompiuterizuota informacijos sistemų inžinerija	7	2016.10.	2017.04.14	8
Sistemų analizės technologijos	7	2017.09.	2016.10.20	8

Conferences

- *DATAMSS2015 - 7th International Workshop Data Analysis Methods for Software. Date: gruodžio 3-5 d., 2015 m., Druskininkai.*

Publications

- *Veitaitė I., Lopata A., N.Žemaitytė (2016) Enterprise Model based UML Interaction Overview Model Generation Proces. 19th International Conference on Business Information Systems, BIS2019 International Workshop, Series: Lecture Notes in Business Information Processing. ISBN 978-3-319-26762-3*

The text of the publication will be presented in the Appendix No 1.

Activities in the Kaunas faculty

Consultation of the 4th course bachelor student.

3 References

1. [1] Mizoguchi, R. and Bourdeau, J. (2000), "Using ontological engineering to overcome common AI-ED problems", International Journal of Artificial Intelligence in Education, vol. 11, pp. 107-121.
2. [2] Boris Motik, Alexander Maedche, and Raphael Volz. A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications
3. [3] Duineveld, A. and Stoter, R. and Weiden, M. and Kenepa, B. and Benjamins, V. (2000), "Wondertools? A comparative study of ontological engineering tools", International Journal of Human-Computer Studies, vol. 52, no. 6, pp. 1111-1113.
4. [4] Helmut Meisel, Ernesto Compatangelo (2004) An ontology-based architecture for the design of knowledge bases in Intelligent Instructional Systems. Department of Computing Science, University of Aberdeen AB24 3UE Scotland, UK
5. [5] Gasevic, D.V., Djuric, D.O. and Devedzic, V.B. Bridging MDA and OWL Ontologies. Journal of Web Engineering, Vol. 4, No. 2, pp. 18-143, 2005
6. [6] D.Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/
7. [7] Bhatt, M. and Taniar, D. A Distributed Approach to Sub-Ontology Extraction. Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04), IEEE, 2004.
8. [8] Colomb, R.M., Raymond, K., Hart, L., Emery, P., Chang, D., Kendall, E. and Frankel, D. Draft Version 2.0: The Object Management Group Ontology Definition Metamodel, 2004
9. [9] Cranfield, S. and Purvis, M. UML as an Ontology Modeling Language, In Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence, IJCAI-99, 1999
10. [10] Baclawski, K., Kokar, M., Kogut, P., Hart, L., Smith, J., Holmes, W., Letkowski, J., Aronson, M. Extending UML to Support Ontology Engineering for the Semantic Web. UML 2001, Toronto, CA, Oct 2001.

11. [11] Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuiness, D.L. Patel-Schneider, P.F. and Stein, L.A. OWL Web Ontology Language Reference. W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-ref-20040210/, 2004
12. [12] Gruber T.R. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2): 199-220, 1993
13. [13] Pretorius, A.J. Visual Analysis for Ontology Engineering, Journal of Visual Languages and Computing, Vol. 16, pp. 359-381, 2005
14. [14] Pisanelli, D. M., Gangemi, A., Steve, G. Ontologies and Information Systems: the Marriage of the Century? In the International Workshop on Governmental Methodology for Software Providence (LYEE02), 2002.
15. [16] Hart, L., Emery, P., Colomb, R. M., Raymond, K., Chang, D., Ye, Y., Kendall, E. and Dutra, M. Usage Scenarios and Goals for Ontology Definition Metamodel. Lecture Notes in Computer Science, Vol. 3306, pp 596, Springer Berlin, 2004
16. [17] M. Nazir Ahmad, Robert M. Mohd Taib Abd Wahid (2007) A Comparison of Ontology Servers
17. [18] IDEAS: A Gap Analysis, www.ideas-roapmap.net, 2003.
18. [19] D. S. Frankel: Model Driven Architecture – Applying MDA to Enterprise Computing, Wiley, 2003.
19. [20] O. Lassila and R.R.Swick. Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/
20. [21]. T. Gardner, C. Griffin, J. Koehler, R. Hauser: A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard, Meta-Modelling for MDA Workshop, 2003.
21. [22] OMG: Model Driven Architecture (MDA), OMG ormsc/2001-07-01, 2001
22. [23] D. Djurić, D. Gašević, V. Devedžić: Ontology Modeling and MDA, Journal of Object Technology, ETH Zurich, 2005.
23. [24] Stephan Roser. Advisor: Bernhard Bauer (2006) Ontology based model transformations. Progamming of Distributed Systems Institute of Computer Science, University of Augsburg, Germany.
24. [25] B. Elvesæter, A. Hahn, A-J. Berre, T. Neple: Towards an Interoperability Framework for Model-Driven Development of Software Systems, First

International Conference on Interoperability of Enterprise Software and Applications, Geneva Switzerland, 2004.

25. [26] Julita Bermejo Alonso (2006) Ontology – based Software Engineering. ASLab-ICEA-R-2006-016. URL: http://www.aslab.org/documents/ASLab-ICEA-2006-016.pdf
26. [27] Force, S. E. T. (2001). Ontology driven architectures and potential uses of the semantic web in systems and software engineering.
27. [28] Tanasescu, V. (2005). An ontology–driven life–event portal. Master, Computer Science.
28. [29] Ruiz, F. and Hilera, J. (2006). Ontologies for Software Engineering and Software Technology, chapter Using Ontologies in Software Engineering and Technology, pages 49–102. Springer-Verlag Berlin Heidelberg.
29. [30] Hesse, W. (2005). Ontologies in the software engineering process. In Lenz, R., editor, Proceedings of Tagungsband Workshop on Enterprise Application Integration (EAI2005), Berlin, Germany. GITO–Verlag.
30. [31] Abran, A., Cuadrado, J., Garc´ıa-Barriocanal, E., Mendes, O., S´anchez-Alonso, S., and Sicilia, M. (2006). Ontologies for Software Engineering and Software Technology, chapter Engineering the Ontology for the SWEBOK: Issues and Techniques, pages 103–121. Springer-Verlag Berlin Heidelberg.
31. [32] Liao, L., Qu, Y., and Leung, H. (2005). A software process ontology and its application. In Proceedings of Workshop on Sematic Web Enable Software Engineering (SWESE), Galway, Ireland.
32. [33] Grûber, T.: A translation approach to portable ontology specification. Knowledge Acquisition 5(2) (1993) 199-220.
33. [34] Allemang, D., Hendler, J.A.: Semantic web for the working ontologist: Modeling in RDF, RDFS and OWL. Elsevier, Amsterdam (2008).
34. [35] Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004).
35. [36] Smith, M., Welty, C., McGuiness, D.: Owl web ontology language guide. Recommendation W3C 2(1) (2004).
36. [37] Verónica Castañeda, Luciana Ballejos, Ma. Laura Caliusco, Ma. Rosa Galli (2010) The Use of Ontologies in Requirements Engineering. Vol. 10 Issue 6 (Ver 1.0) November

37. [38] Petr Kroha, Jose Emilio Labra Gayo (2008) Using Semantic Web Technology in Requirements Specifications
38. [39] Katja Siegemund, Edward J. Thomas, Yuting Zhao, and Uwe Assmann (2010) Towards Ontology-driven Requirements Engineering
39. [40] V. Castaneda, L. Ballejos, L. Caliusco, R. Galli (2010) The use of Ontlogies in Requirements Engineering. Vol. 10 Issue 6 (Ver 1.0) November 2010. Global journal of researches in engineering.
40. [41] J. Trinkunas, O. Vasilecas (2009) Ontology transformation: from requirements to conceptual model. Scientific Papers, Universit y of Latvia, 2009. Vol. 751 Computer Science and Information Technologies
41. [42] Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering. Springer, New York. USA. (2004).
42. [50] Eric S. K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE '97, pages 226–235, Washington, DC, USA, 1997. IEEE Computer Society.
43. [52] Stefan Farfeleder, Thomas Moser, Andreas Krall, Tor Stålhane, Inah Omoronyia, and Herbert Zojer. Ontology-driven Guidance for Requirements Elicitation. In Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part II, ESWC'11, pages 212–226, Berlin, Heidelberg, 2011. Springer-Verlag.
44. [53] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: an Environment for Goal-driven Requirements Engineering. In Proceedings of the 19th international conference on Software engineering, ICSE '97, pages 612–613, New York, NY, USA, 1997. ACM.
45. [54] http://stackoverflow.com/questions/1740341/what-is-the-difference-between-rdf-and-owl
46. [56] https://www.slideshare.net/rlovinger/rdf-and-owl
47. [58] Ivan J. Jureta, John Mylopoulos, and Stéphane Faulkner. A Core Ontology for Requirements. Appl. Ontol., 4(3-4):169–244, 2009.
48. [59] Haruhiko Kaiya and Motoshi Saeki. Ontology Based Requirements Analysis:

49. Lightweight Semantic Processing Approach. In Proc. Fifth International Conference on Quality Software (QSIC 2005), 2005.
50. [60] Thomas Riechert, Kim Lauenroth, and Jens Lehmann. Semantisch unterstütztes Requirements Engineering. In Proceedings of the SABRE-07 SoftWiki Workshop, 2007.
51. [62] Leonid Kof. Natural Language Processing For Requirements Engineering Applicability.
52. In Proceedings of the Workshops, page 2004, 2004.
53. [63] Yang Ying-ying, Li Zong-yon, and Wang Zhi-xue. Domain Knowledge Consistency Checking for Ontology-Based Requirement Engineering. In CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering, pages 302–305, Washington, DC, USA, 2008. IEEE Computer Society.
54. [64] Xuefeng Zhu. Inconsistency Measurement of Software Requirements Specifications: An Ontology-Based Approach. In ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, pages 402–410, Washington, DC, USA, 2005. IEEE Computer Society.
55. [65] M. Kassab, O. Ormandjieva, and M. Daneva. An Ontology Based Approach to Non-functional Requirements Conceptualization. Software Engineering Advances, International Conference on, 0:299–308, 2009.
56. [66] M. Kossmann, R. Wong, M. Odeh, and A. Gillies. Ontology-driven Requirements Engineering: Building the OntoREM Meta Model. In Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on, pages 1 – 6, 2008.
57. Šaltinis: [67]
58. [72] Su, X., Ilebrekke, L.: A Comparative Study of Ontology Languages and Tools. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 761–765. Springer, Heidelberg (2002)
59. [73] D. Kalibaitienė, O. Vasilecas, Survey on Ontology languages, 2011, Vilnius Gediminas Technical University. DOI: 10.1007/978-3-642-24511-4_10 · Source: DBLP

60. [74] J. Trinkūnas, Research on conceptual data modelling using Ontology, (2010), Doctoral dissertation, VGTU, Vilnius. ISBN 978-9955-28-589-2

61. [75] T. Slimani (2015) Ontology Development: A Comparing Study on Tools, Languages and Formalisms. Indian Journal of Science and Technology, Vol 8(24), DOI: 10.17485/ijst/2015/v8i34/54249. ISSN (Online) : 0974-5645

62. [76] Veitaitė I., Lopata A., Žemaitytė N. (2016) Enterprise Model based UML Interaction Overview Model Generation Proces. 19th International Conference on Business Information Systems, BIS2019 International Workshop, Series: Lecture Notes in Business Information Processing. ISBN 978-3-319-26762-3.

63. [77] SEBoK: http://sebokwiki.org/wiki/System_Requirements

1. https://www.w3.org/TR/2012/REC-owl2-primer-20121211/#OWL_Syntaxes
2. https://www.w3.org/OWL/
3. https://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

4 Appendixes

Appendix No 1

Ontologies and Enterprise Modelling based UML Models Generation Process (UML Interaction Overview Case)

Audrius Lopata, Ilona Veitaitė, Neringa Žemaitytė

Vilnius University, Kaunas Faculty of Humanities, Department of Informatics
Muitinės g. 8, LT-44280, Kaunas, Lithuania

Audrius.Lopata@khf.vu.lt, Ilona.Veitaite@khf.vu.lt,
Neringa.Zemaityte@khf.vu.lt,

Abstract. The main scope of the research is to analyse Unified Modelling Language (UML) models generation process from Enterprise Model (EM) in Information Systems (IS) development process by using knowledge-based subsystem. The knowledge-based subsystem is proposed as an additional computer aided software engineering (CASE) tool component to avoid IS development process based on empirics. For comprehensible perception there is also presented relation between EM and ontologies and its use in generation process.
As the result of this part of research transformation algorithms are presented and described. These algorithms are capable of whole UML models elements generation from Enterprise Model. Example of UML Interaction Overview model generation illustrates full process.

Keywords: Enterprise Modelling, Knowledge-based, IS engineering, UML, CASE, Ontology, Interaction Overview model.

1 Introduction

In a modern world software development and software applications are becoming more complex and demanding. Developers, analysts, engineers, researchers are creating and seeking for new techniques and procedures to streamline software engineering processes to ensure shorter development time and reduce costs by re-using different components. The development of software systems is a complex activity which may imply the participation of people and machines (distributed or not). Therefore, different stakeholders, heterogeneity and new software features make software development a heavily knowledge-based process [1, 11].

In a modern day enterprise engineering, it is paramount that Enterprise Models are grounded in a well-defined, agreed-upon Enterprise Architecture that captures the essentials of the business, IT, and its evolution. Enterprise architectures typically contain different views (e.g. Business, Information, Process, Application, Technical) on the enterprise that are developed by distinct stakeholders with a different background and knowledge of the business. Consequently, the developed Enterprise Models that populate these views are hard to integrate. A possible solution for this integration problem is using a shared terminology during the development of these different views [2]. Such explicit formal representations, often materialized in the form of ontology – in a business context called an enterprise-specific ontology – provide a myriad of advantages. Ontologies are shared views of domains. They provide conceptualizations that are agreed upon by participants in collaborative action and decision making. The explicit existence of such shared perspectives makes it possible for both people and programs to collaborate by ensuring that everybody makes the same distinctions and uses the same terms with the same meaning [19]. On an intra-organizational level, they ensure model re-usability, compatibility and interoperability, and form an excellent basis for enterprise-supporting IT tools, such as Enterprise Resource Planning (ERP) systems, business intelligence (BI) tools or information systems (IS), for which they serve as common terminology. On an inter-organizational level, they facilitate interoperability, cooperation and integration by allowing formal mappings between, and alignment of separately developed Enterprise Models [12].

2 Enterprise Modelling and Ontologies relation

An Enterprise Model is a computational representation of the structure, activities, processes, information, resources, people, behaviour, goals and constraints of a business, government, or other enterprise. It can be both descriptive and definitional - spanning what is and what should be. The role of an Enterprise Model is to achieve model-driven enterprise design, analysis and operation [6, 19]. Enterprise Modelling is an activity where an integrated and commonly shared model of an enterprise is created [7, 12, 28]. The resulting Enterprise Model comprises several sub-models, each

representing one specific aspect of the enterprise, and each modelled using an appropriate modelling language for the task at hand. For example, the Enterprise Model may contain processes modelled in BPMN, data modelled in ER and goals modelled in n*. The Enterprise Model is thus developed by several enterprise engineers, and aggregates all information about the enterprise. As a result, Enterprise Models without homogenized underlying vocabulary suffer interoperability and integration problems [12, 25]. An Enterprise Model can be developed for single or more different purposes. Few Enterprise Modelling formal purposes are presented [3, 21]:

1. To capitalize enterprise knowledge and know how.
2. To illustrate relations and dependencies within the enterprise and with other enterprises, to achieve better control and management over all aspects.
3. To provide support to business process re-engineering.
4. To get a common and complete understanding of the enterprise.
5. To improve information management across organizational and application system boundaries and provide a common means for communication throughout the organization. Rationalize and secure information flows.
6. To provide operative support for daily work at all levels in the enterprise from top to bottom.
7. To control, co-ordinate and monitor some parts of the enterprise.
8. To provide support for decision making.
9. To provide support the design of new parts of the enterprise.
10. To simulate processes.

Ontology is a discipline rooted in philosophy and formal logic, introduced by the Artificial Intelligence community in the 1980s to describe real world concepts that are independent of specific applications. Over the past two decades, knowledge representation methodologies and technologies have subsequently been used in other branches of computing where there is a need to represent and share contextual knowledge independently of applications [23].

Ontology engineering is a filiation of knowledge engineering that studies the methods and methodologies for building ontologies. In the domain of enterprise architecture, ontology is an outline or a schema used to structure objects, their attributes and relationships in a consistent manner. As in Enterprise Modelling, ontology can be composed of other ontologies. The purpose of ontologies in Enterprise Modelling is to formalize and establish the shared understanding, reuse, assimilation and dissemination of information across all organizations and departments within an enterprise. Also, an ontology enables integration of the various functions and processes which take place in an enterprise [10].

Using ontologies in Enterprise Modelling offers several advantages. Ontologies ensure clarity, consistency, and structure to a model. They promote efficient model definition and analysis. Generic enterprise ontologies allow for reusability and automation of components. A common ontology allows to ensure shared understanding, clearer communication, and more effective coordination among the various divisions of an enterprise. These lead to more efficient production and flexibility within the enterprise [24].

3 Transformation Algorithm

The computerized IS engineering specific methods are developed based on common requirements, which systematize the selected methodology. Computerized knowledge-based IS engineering project management basis is CASE system knowledge-based subsystem. CASE system's knowledge-based subsystem's core component is

knowledge base, which essential elements are enterprise meta–model specification and Enterprise Model for certain problem domain [4, 8, 25]. Knowledge-based subsystem is one more active participant of IS engineering process beside analyst, whose purpose is to verify results of IS life cycle phases [5, 9].
Knowledge-based CASE systems holding substantial components, which organize knowledge: knowledge-based subsystem's knowledge base, which essential elements are enterprise meta–model specification and Enterprise Model for certain problem domain [7, 13, 16].

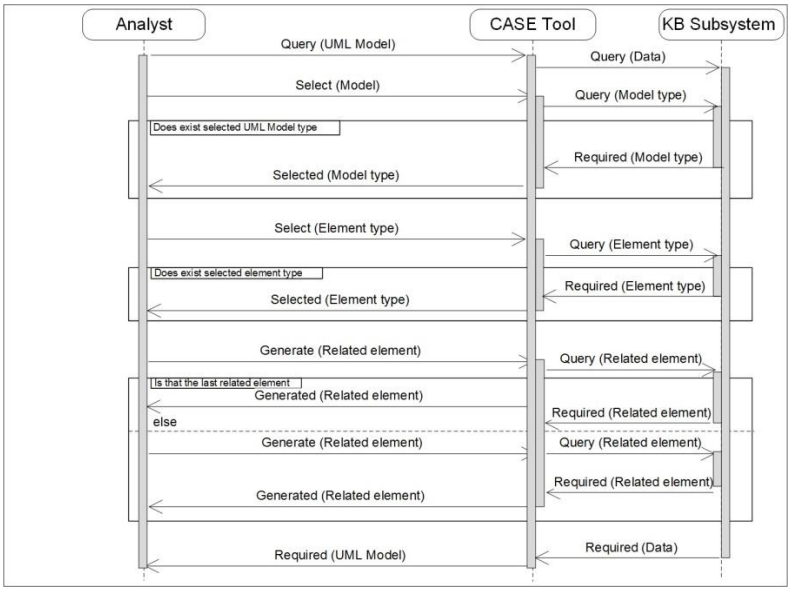


Fig. 1. Knowledge-based subsystem connection to the Enterprise Model and enterprise meta–model inside CASE tool presented as Sequence diagram

Information system design methods indicate the continuance of systems engineering actions, i.e. how, in what order and what UML models to use in the design process and how to fulfil the process. Association between UML models and Enterprise Model is realized through the transformation algorithms [14, 15].

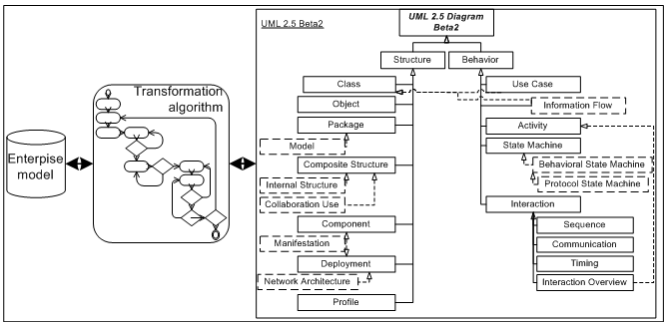


Fig. 2. UML models generation by using the transformation algorithm [22]

Enterprise Model as organization's knowledge repository enables generate UML models with the help of transformation algorithms. Enterprise meta-model holds essential elements of business modelling methodologies and techniques, which ensures a proper UML models generation process [17, 18, 20].
Presently, used CASE system's Enterprise Models constitution is not verified by formalized criteria. Enterprise Models have been formed in compliance with the

notations. However, their composition has not been proved by the characteristics of the specific domain area [27, 28].
In IS engineering all design models are fulfilled on the basis of the empirical expert experience. Experts, who participate in the IS development process, do not gain enough knowledge, and process implementation in requirements analysis and specification phases can take a too long time. Enterprise meta–model contains essential elements of business modelling methodologies and techniques, which insures a suitable UML diagrams generation process [27, 28].

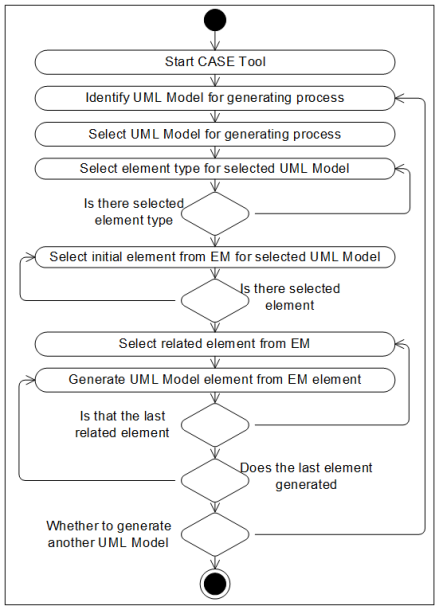


Fig. 3. Transformation Algorithm of EM based UML model generation process

Figure 3 presents top level transformation algorithm for Enterprise meta–model based UML models generating process. Main steps for generating process are identifying and selecting UML model for generating process, identifying starting elements for the selected UML model and selecting all related elements, reflecting Enterprise Model elements to UML model elements and generating the selected UML model [22, 27, 28].

4 UML Interaction Overview Model Transformation

UML Interaction Overview diagram determines interactions through a variant of activity diagrams in a manner that maintains overview of the control flow. Interaction Overview model concentrate on the overview of the flow of control where the nodes are interactions or interaction uses. The lifelines and the messages do not perform at this overview level. UML Interaction Overview model combines elements from activity and interaction diagrams [22]:

- the following elements of the activity diagrams could be used on the Interaction Overview diagrams: initial node, flow final node, activity final node, decision node, merge node, fork node, join node;
- the following elements of the interaction diagrams could be used on the Interaction Overview diagrams: interaction, interaction use, duration constraint, time constraint.

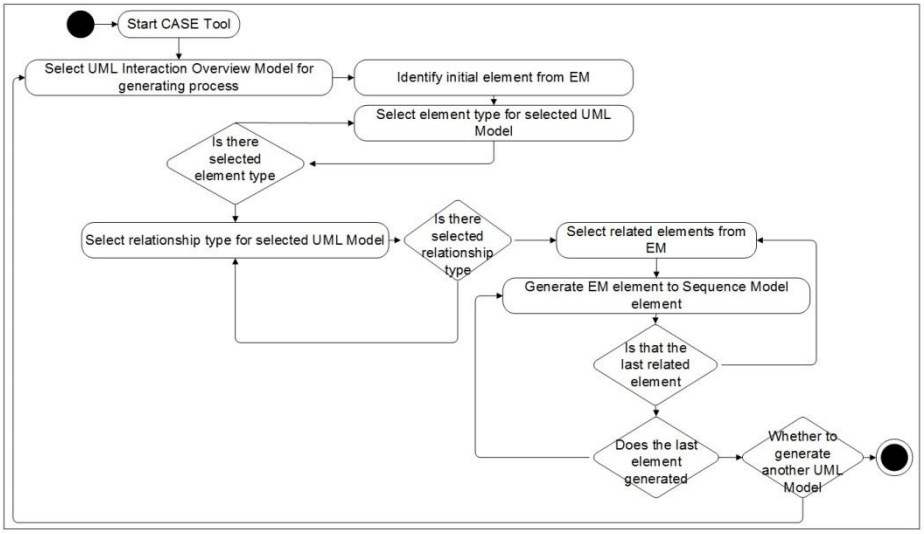


Fig. 4. Transformation Algorithm of EM based UML Interaction Overview model generation process

Main steps of UML Interaction Overview model generation from Enterprise Model transformation algorithm are: selecting Interaction Overview model for generating process, identifying initial element, selecting element's type for chosen model, selecting related model elements and generating model.

Table 1 presents UML Interaction Overview model elements generated from Enterprise Model. Frame as Interaction model element is generated from EM Actor element, Interaction Use as Interaction model element is generated from EM Information Activity, Initial Node, Decision Node, Merge Node, Final Node as Activity model elements are generated from EM Business Rules elements and Decision Guard as Activity model element is generated from EM Information Flow element.

Table 1. EM and Online Service Ordering UML Interaction Overview model elements

UML Interaction Overview model / **EM**		Frame (Interaction model)	Interaction Use (Interaction model)	Initial Node (Activity model)	Decision Node (Activity model)	Merge Node (Activity model)	Final Node (Activity model)	Decision Guard (Activity model)
Actor		+						
Event								
Process	Material Input							
	Material							
Function	Business Rules			+	+	+	+	
	Information							+
	Information		+					

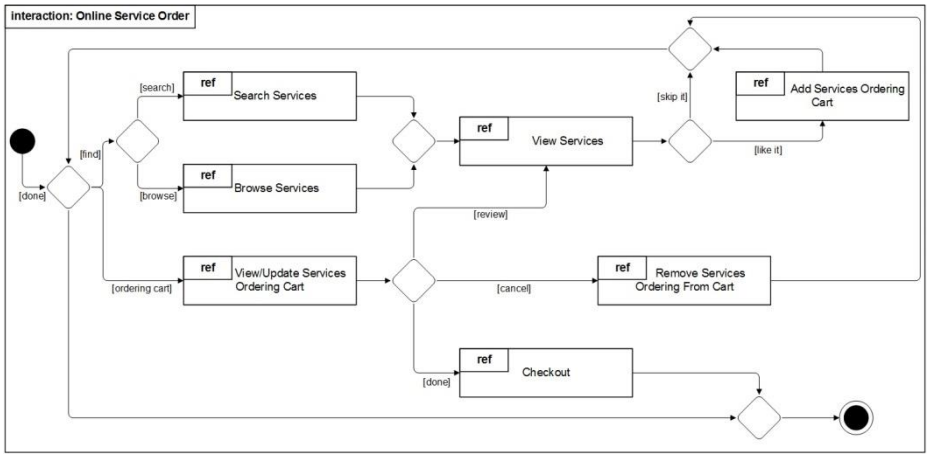


Fig. 5. UML Interaction Overview model example: Online Service Order

Figure (Fig. 5) presents an example of UML Interaction Overview model. The necessary elements through transformation algorithms were received from CASE tool's knowledge-based subsystem's Enterprise Model, where all knowledge of subject area is stored. In this figure it is clearly seen all necessary UML Interaction Overview model elements generated from Enterprise Model.

5 Conclusions

Computer aided IS engineering is based on empiric and IS development life cycle stages are fulfilled on the basis of the expert's experience. A large part of the CASE tools design models are generated only partially, and complete realization is possible only non-automatic and with experts participation. Today IS engineering should be based on knowledge. In this way, knowledge-based IS engineering computerized IS development activities are executed using the subject area knowledge, which is stored in the knowledge base of CASE tool repository.

In order to decrease the influence of empirical factors on IS development process, the decision was made to use knowledge-based IS engineering approach. The main advantage of this approach is the possibility to validate specified data stored in EM against formal criteria, in that way decreasing the possible issues and ensuring more effective IS development process compared to classical IS development methods.

Using ontologies in Enterprise Modelling offers several advantages. Ontologies ensure clarity, consistency, and structure to a model. They promote efficient model definition and analysis. Generic enterprise ontologies allow for reusability of and automation of components. Because ontologies are schemes or outlines, the use of ontologies does not insure proper Enterprise Model definition and analysis. Ontologies are limited by how they are defined and fulfilled. Ontology not always includes ability to cover all of the aspects of what is being modelled.

The paper deals with the generation process of UML models from EM options. Every element of UML model can be generated from the EM using CASE Tool knowledge base's subsystem and transformation algorithms. Method of UML model generation process from EM could implement full knowledge-based IS development cycle design stage. This is partially established by the example of online service ordering presented as UML Interaction Overview model elements generation.

6 References

11. Alonso Julita B. (2006). Ontology-based Software Engineering, Engineering Support for Autonomous Systems. ASLab-ICEA-R-2006-016 v 0.1 Draft of 2006-11-15.
12. Bera, P., Burton-Jones, A., Wand, Y. (2011) Guidelines for Designing Visual Ontologies to Support Knowledge Identification. Mis Quarterly 35, 883–908
13. Brathaug T. A.&Evjen T.Å. (1996) Enterprise Modeling. SINTEF, Trondheim
14. Butleris, R., Lopata, A., Ambraziunas, M., Veitaitė, I., Masteika S. (2015) SysML and UML models usage in knowledge based MDA process. Elektronika ir elektrotechnika. Vol 21, No 2 (2015) pp. 50-57 Print ISSN: 1392-1215, Online ISSN: 2029-5731
15. IEEE Computer Society (2014) Guide to the Software Engineering Body of Knowledge SWEBOK. Version 3.0. Paperback ISBN-13: 978-0-7695-5166-1
16. Gudas S. Enterprise knowledge modelling: Domains and aspects. Technological and economic development of Economy. Baltic Journal on Sustainability 281–293, 2009
17. Gudas S., Architecture of Knowledge-Based Enterprise Management Systems: a Control View, Proceedings of the 13th world multiconference on systemics, cybernetics and informatics (WMSCI2009),) July 10 – 13, 2009, Orlando, Florida, USA, Vol. III, p.161-266 ISBN -10: 1-9934272-61-2 (Volume III).ISBN -13: 978-1-9934272-61-9 (Volume III)
18. Gudas S., (2012) Informacijos sistemų inžinerijos teorijos pagrindai. Vilniaus universiteto leidykla ISBN 978-609-459-075-7
19. Gudas s., Lopata A., (2007) Meta-Model Based Development Of Use Case Model For Business Function. Information Technology And Control, ISSN 1392 – 124X 2007, Vol.36, No.3
20. Fadel, G., Fox, M., Gruninger, M. (1994). A Generic Enterprise Resource Ontology. In: Proceedings of the 3rd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. p. 117-128
21. Force, S. E. T. (2001). Ontology driven architectures and potential uses of the semantic web in systems and software engineering.
22. Frederik Gailly, Sven Casteleyn, Ndejda Alkhaldi (2013) On the Symbiosis between Enterprise Modelling and Ontology Engineering. Ghent University, Universitat Jaume I, Vrije Universiteit Brussel, DOI: 10.1007/978-3-642-41924-9_42.
23. Lopata A. Disertacija. Veiklos modeliu grindžiamas kompiuterizuotas funkcinių vartotojo reikalavimų specifikavimo metodas. 2004
24. Lopata, A., Veitaite, I., Gudas, S., Butleris, R. (2014) CASE Tool Component – Knowledge-based Subsystem. UML Diagrams Generation Process. Transformations in Business & Economics, Vol. 13, No 2B (32B) pp. 676-696 ISSN: 1648 - 4460
25. Lopata, A., Veitaite, I. (2013) UML Diagrams Generation Process by Using Knowledge-Based Subsystem. Tarptautinė konferencija „15th International Conference on Business Information Systems“, BIS2013 (5th Workshop on Applications of Knowledge-Based Technologies in Business (AKTB 2013)).
26. Lopata A., Ambraziūnas M., Gudas S., Butleris R. (2012) „The Main Principles of Knowledge-Based Information Systems Engineering“, Electronics and Electrical Engineering, Vol. 11, No 1 (25), pp. 99-102, ISSN 2029-5731.
27. Lopata A., Ambraziūnas M., Gudas S. (2012) “Knowledge Based MDA Requirements Specification and Validation Technique”, Transformations in Business & Economics, Vol. 11, No. 1 (25), pp. 248-261.
28. Lopata, A., Ambraziūnas, M., Gudas, S. Knowledge-based MDA requirements specification and validation technique. Transformations in Business & Economics, 2012, 11(1(25)), 248-260. ISSN 1648-4460.
29. Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, and Jinxin Lin (1998), An Organization Ontology for Enterprise Modelling. MIT Press Cambridge, MA, USA p. 131-152, ISBN:0-262-66108-X
30. Morkevicius A., Gudas S. (2011) “Enterprise Knowledge Based Software Requirements Elicitation”, Information Technology and Control, Vol. 40, No 3, pp. 181-190, 1392 – 124X
31. Nadeem Ahmed Khan (2011) Transformation of Enterprise Model to Enterprise Ontology. Master Thesis, Jonkoping, Sweden.
32. OMG UML (2012) Unified Modelling Language version 2.5. Unified Modelling// http://www.omg.org/spec/UML/2.5/Beta2/

33. OMG ODM (2014) OMG Formal Versions of Ontology Definition Metamodel// http://www.omg.org/spec/ODM/1.1
34. Ostie James K. (1996). An Introduction to Enterprise Modeling and Simulation
35. Perjons, E (2011) Model-Driven Process Design. Aligning Value Networks, Enterprise Goals, Services and IT Systems. Department of Computer and Systems Sciences, Stockholm University. Sweden by US-AB, Stockholm ISBN 978-91-7447-249-3
36. Stirna, J., Persson, A., Sandkuhl, K. (2007) Participative Enterprise Modeling: Experiences and Recommendations. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 546–560. Springer, Heidelberg
37. Veitaitė I., Lopata A. (2015) Additional Knowledge Based MOF Architecture Layer for UML Models Generation Process. 18th International Conference on Business Information Systems, BIS2015 International Workshop, Series: Lecture Notes in Business Information Processing.
38. Veitaite I., Ambraziunas M., Lopata A. (2014) Enterprise Model and ISO Standards Based Informations System's Development Process. 16th International Conference on Business Information Systems, BIS2014 International Workshop, Larnaca, Cyprus, May 22-23, 2014, Series: Lecture Notes in Business Information Processing.
39. Vernadat, F. (2002) UEML: towards a unified Enterprise Modelling language. International Journal of Production Research 40, 4309–4321