An overview of Requirements Documentation and Specification

Hong Sun

November 23, 2020

Abstract

This report summarizes different approaches in requirements documentation and specification. Also it tries to cover some new advances and open questions.

1 Introduction

Requirements Engineering (RE) is the process of defining, documenting, and maintaining requirements [1]

Requirements engineering process is composed of several steps, including domain understanding, requirements elicitation, evaluation and agreement, specification and documentation. [2]

Requirements documentation is a phase in which the results from requirements elicitation and evaluation are precisely specified and documented. The input of requirements documentation is a set of agreed statements, for example general objectives, system requirements and environmental assumptions. The output of this phase is the requirements documents, which may need future updates. [2].

2 Approaches in Requirements Documentation

Requirements could be documented in different ways. In the early stage, requirements elicited from the clients may be written in natural languages. Based on proper interpretation, the requirements may be expressed in diagrams and even formal languages afterwards.

2.1 Documentation in natural language

2.1.1 Free language documentation

According to [3], a majority of documents available for requirements analysis are provided by the user or are obtained by interviews. Moreover,

- 71.8% of these documents are written in common natural language,
- 15.9% of these documents are written in structured natural language, and
- only 5.3% of these documents are written in formalized language.

Due to the imprecise nature of human languages, there may be many potential problems in the requirements. Some of the potential problems are listed below.

- Terminology clash. The same concept is given different names in different statements. For example, "participant" and "attendant" actually refer to the same entity. Terminology clash may cause confusion in the communication.
- Conflict. There are some statements that cannot be satisfied at the same time. When there is a conflict, we need to solve it to make the requirements self-consistent.
- Redundancy. Some statements can be satisfied by other statements.

2.1.2 Structured language documentation

In order to mitigate the problems in free-style natural languages, structured natural languages can be used to generate clearer documents. There are two kinds of rules in the application of structured natural languages. Local rules state how statements should be written in natural language, while global rules state how the requirements document should be organized [2].

Examples of local rules from [2] are shown below,

- A single sentence should contain no more than one requirement, assumption or domain property.
- Keep sentences short.
- Use decision table for combinations of conditions.

Regarding the global rules, IEEE provides some guidelines and templates for Requirements Specifications (RS).

IEEE Standard 29148 [4] suggests using specific keywords and terms that signal the presence of a requirement. A common approach is to stipulate the following:

- Requirements are mandatory binding provisions and use 'shall'.
- Statements of fact, futurity, or a declaration of purpose are non-mandatory, non-binding provisions and use 'will'. 'Will' can also be used to establish context or limitations of use. However, 'will' can be construed as legally binding, so it is best to avoid using it for requirements.
- Preferences or goals are desired, non-mandatory, non-binding provisions and use 'should'.
- Suggestions or allowances are non-mandatory, non-binding provisions and use 'may'.
- Non-requirements, such as descriptive text, use verbs such as 'are', 'is', and 'was'. It is best to avoid using the term 'must', due to potential misinterpretation as a requirement.
- Use positive statements and avoid negative requirements such as 'shall not'.
- Use active voice: avoid using passive voice, such as 'shall be able to select'.

[4] also includes several templates for different requirements specification, including **Stakeholder Requirements Specification (StRS)**, **System Requirements Specification (SyRS)** and **Software Requirements Specification (SRS)**.

2.1.3 Applying NLP techniques

The close relationship between natural language (NL) and requirements has been a source of inspiration for researchers to seek to apply natural language processing (NLP) techniques and tools to processing requirements texts. [5]

Kevin Ryan argued that RE cannot be fully automated by NLP [5]. His argument is based on the following observations

- Some requirements are expressed in diagrams, not in textual form.
- Some requirements cannot be formalized, for example "user-friendly".

- Showing the conformance between a requirements document and the clients' needs is actually a social process.

Despite the above restrictions, based on the past and current empirical evidence, we can safely assume that NL will continue to serve as the *lingua franca* for requirements in the future as well. [6]

To promote the research of NLP in RE, NLP4RE (NLP for Requirements Engineering) workshop was launched in 2018. [7]

Though there is a view that NLP can still help with some aspects of RE, numerous challenges remain. The following challenges in NLP4RE are presented by [7].

- Resource Availability. The effectiveness of NLP depends on existing resources, including high-quality data corpora, validation metrics and workflows.

- Context Adaptation. This is about customizing NLP algorithms in different context. For example, a NLP application trained on English corpora may not work effectively on other languages. Also, if the training corpus is for a particular domain (like healthcare), there is no guarantee the algorithms will be effective on security critical applications.

- Player Cooperation. To successfully apply NLP into practice, researchers must cooperate closely with industries, NLP experts and tool vendors.

In analyzing the literature of NLP in RE, I have identified the following trends and focal points.

1. Ambiguity Detection

A context is always needed when we talk about the ambiguity. If two different terms have very similar contexts, there may be a "terminology clash" problem. For example, "attendant" and "participant" are very likely to have similar contexts and to be interchangeable.

Word embedding is an algorithm that maps a word to a numerical vector [8]. In this way, we can calculate the "distance" between two words. A small distance means the two words share similar contexts.

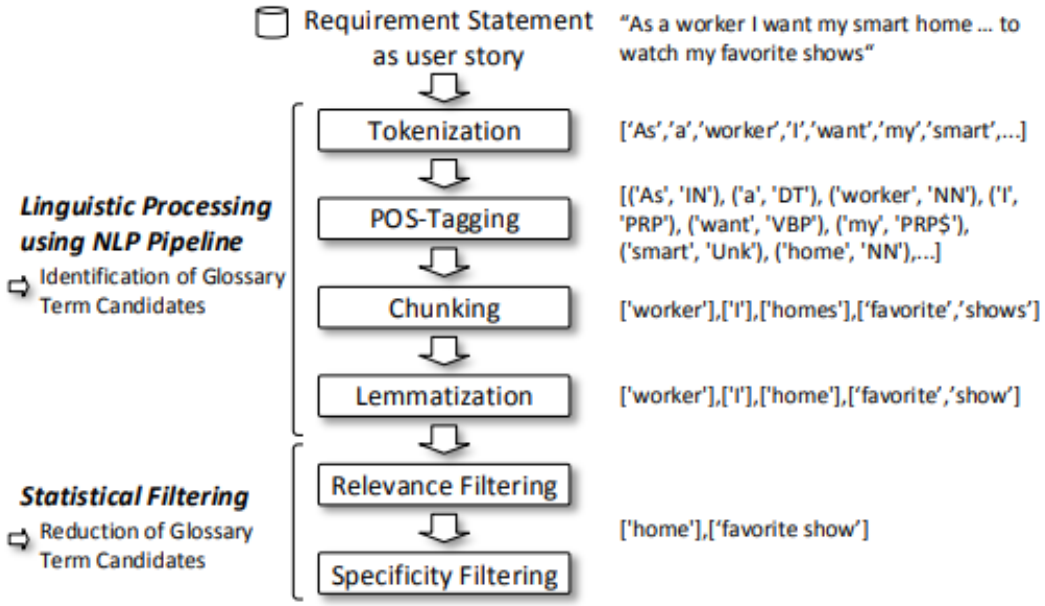
To find potential terminology clashes, we need to calculate the vectors of selected terms in one or more documents. By comparing pair wise distances, we can see which two words may have a clash.

To evaluate the ambiguity of a term across multiple domains, [9] proposes a method based on word embedding. This method calculates an "ambiguity score" for each selected term across different domains. Higher ambiguity score means a higher possibility of ambiguity the corresponding term has.

2. Information Extraction

Extracting useful information, like glossary terms and potential inconsistencies, from requirement documents has been attractive for a long time.

To pick up glossary terms, [10] proposes a hybrid approach which combines linguistic algorithms and statistical methods. The following figure shows the processing pipeline of the proposed approach.



There are still many open problems in NLP4RE. Based on what I have read so far and my personal understanding, the following issues need our attention in the future:

- Measurements. As many approaches have been proposed by researchers, we need some measurements to compare the performance of different approaches. For example, which metrics we can use to compare two algorithms that aim at detecting ambiguities in requirements.

- Public data sets. In order to compare two or more different approaches, we need

to apply them on the same corpora. [11] provides a public data set of requirements documents. But we still need more high-quality corpora.

2.2 Diagrammatic notations

The importance of visual notations, as well as the characteristics of a well-defined visual representation, is discussed in [12]. Visual representations are processed in parallel by the visual system, while textual representations are processed serially by the auditory system. [12]

There are many diagrams that depict the whole system from different perspectives. This section tries to categorize the diagrams according to their focuses.

2.2.1 System scope

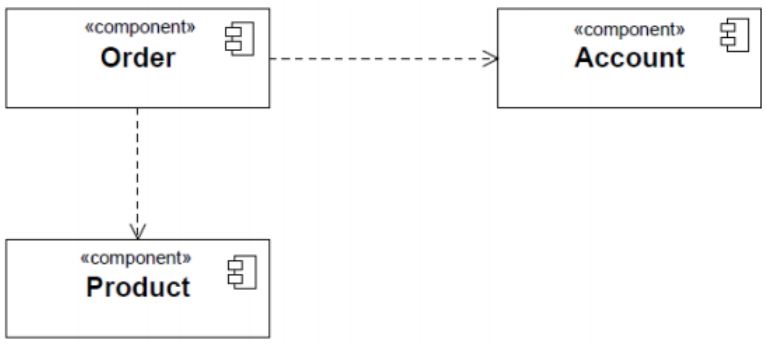
This type of diagrams focuses on the components of a system and their phenomena.

A UML **component diagram** specifies a Component as a modular unit with well-defined Interfaces. [15]

A component has the following features. [15]

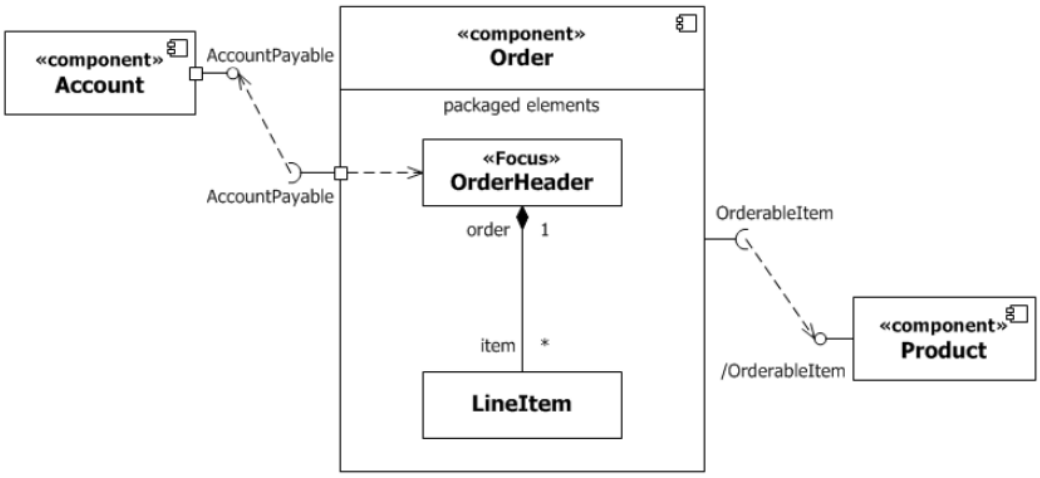
- Self-contained - A Component specifies a formal contract of the services that it provides to its clients and those that it requires from other Components.
- Substitutable - A component can be replaced at design time or run-time by a Component that offers equivalent functionality based on compatibility of its Interfaces.

An overview component diagram usually shows the components and their dependencies. The following figure shows an overview diagram of an E-Business system.



A detailed component diagram may show the interfaces of a component and the internal structure of a component. The following figure is a detailed component of the same E-

Business system.

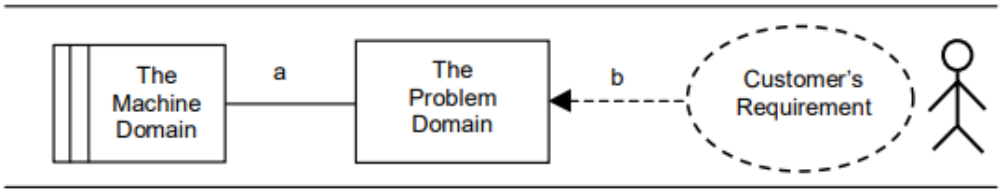


Michael Jackson has proposed a framework to analyze the requirements of a system. The most important principle in this framework is to identify Machine Domain and Problem Domain. [16]

There are three core concepts in this framework. [16]

- Machine Domain - The software and the computers that execute it.
- Problem Domain - The environment(users, stakeholders, etc) that the machine needs to communicate to.
- Requirement - The constraints on the Problem Domain that the Machine must enforce.

The relationship of the Machine, the Problem Domain and the requirement are shown in the following figure.

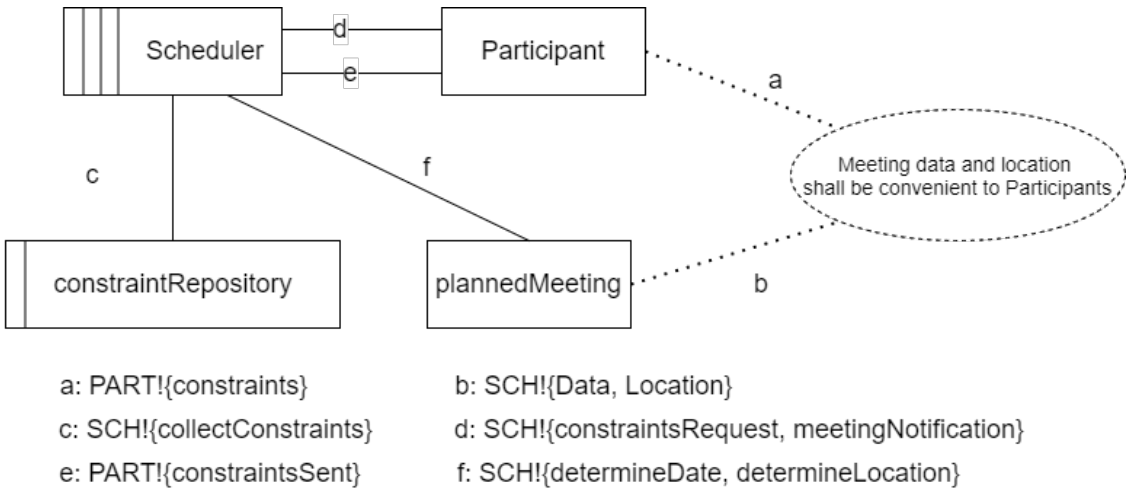


Problem Diagram is proposed by Michael Jackson based on the Machine-Problem Domain framework. [14] Besides the Machine, Problem and Requirement, problem diagram also introduces a Component element. A component is represented by a rectangle with a single stripe. Moreover, there are labels that describe the relationship between different elements.

A label is represented by an exclamation mark after a component's name, appending a set that declares the phenomena controlled by the component.

As an example, let's consider a scheduler that plans meetings according to a set of constraints and participants' favourites.

A problem diagram depicting this scenario is shown below.



2.2.2 Conceptual structures

A system may involve many conceptual items that are related to each other. Entity-Relationship (ER) diagram is designed by Peter Chen to depict the conceptual items and their properties [17].

There are three core elements in an ER diagram [17].

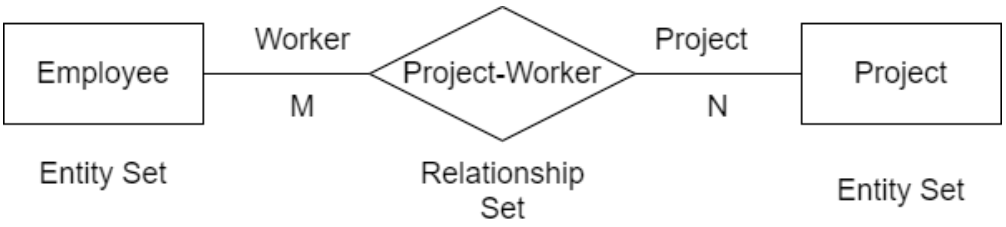
- Entity - An item that can be identified distinctly, for example a specific person or company.
- Relationship - A relationship is an association among entities. For example "father-son" is a relationship between two "person" entities.
- Attribute - Attributes are used to express the information of entities or relationships. For example, a "person" entity have attributes like age, name, address.

According to [17], there are two kinds of relationships.

- Weak relationship - Some entities in the relationship are identified by other relationships.

- Regular relationship - Entities in the relationship are identified by their own attributes.

The following figure shows an example of the ER diagram.



In the above ER diagram, an entity set is represented by a rectangle box, while a relationship set is represented by a diamond-shaped box.

2.2.3 System operations

A system may have multiple users. The users can be categorized into different roles with different operations.

A **Use Case diagram** collects all the operations that an active system component has to perform within a rectangle associated with it. [18]

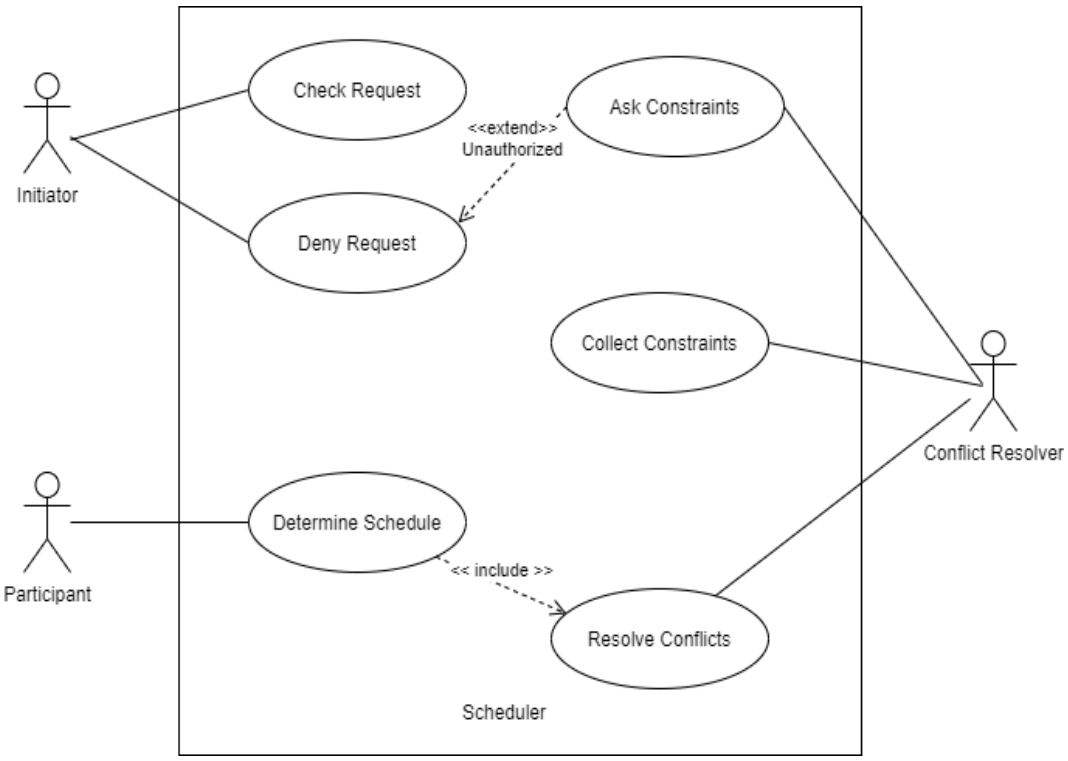
There are three core concepts in a use case diagram. [15]

- Subject - A system under consideration to which the Use Case applies.
- Actor - A role which can perform some operations on the subject.
- Use Case - A specification of behaviour.

There are two kinds of relations between Use Cases. [15]

- Extend - A relationship specifies how and when the behavior defined in the extending Use Case can be inserted into the behavior defined in the extended Use Case.
- Include - Include relationship is intended to be used when there are common parts of the behavior of two or more Use Cases. The common part can be extracted to a separate UseCase, to be included by other Use Cases.

The following figure shows a use case diagram of a meeting scheduling system.



2.3 Formal specification

2.3.1 Formalizing statements

The foundation of formal specification is First-Order logic.

[2] presents the following example of expressing a statement in first-order predicate language.

Statement: When one train follows another train, the minimum distance between them has to be larger than the worst-case stopping distance.

First-order specification: $\forall tr1, tr2 : Train, Following(tr2, tr1) \rightarrow Dist(tr2, tr1) > WCS\text{-}Dist(tr2)$.

In the above specification, the *Following* predicate here returns True if tr2 is following tr1. *Dist* is a function that returns the distance between two trains. *WCS-Dist* is a function that returns the worst-cast stopping distance.

Furthermore, we can use Linear Temporal Logic (LTL) to express some statements.

LTL provides the following temporal connectives.

$\Box$	always in the future
$\Diamond$	some time in the future
$\blacksquare$	always in the past
$\blacklozenge$	some time in the past
$\boldsymbol{W}$	always in the future unless
$\boldsymbol{B}$	always in the past back to
$\boldsymbol{U}$	always in the future until
$\boldsymbol{S}$	always in the past since
$\bigcirc$	in the next state
$\bullet$	in the previous state

2.3.2 Formalizing the UML

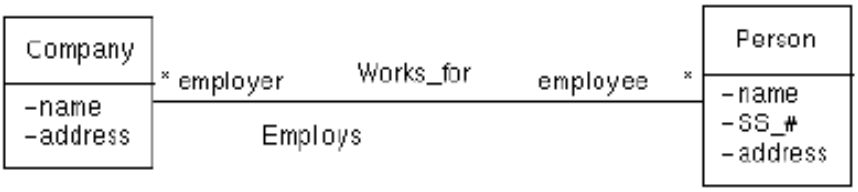
Unified Modeling Language (UML) is widely used as an object-oriented modeling approach. But UML comprises several different notations with no formal semantics attached to the individual diagrams. Therefore, it is not possible to apply rigorous automated analysis or to execute a UML model in order to test its behavior. [22]

[19] argues that rather than attempting to extend traditional **Formal Specification Techniques (FSTs)** with Object-Oriented (OO) concepts, a more workable approach would be to provide a formal basis for the OO modeling notations and concepts.

Generally, there are two approaches in the formalization of UML. The first approach is mapping UML to a formal language such as Z, but there is no universal framework for the rule generation. In the second approach, UML models are first mapped to an intermediate form, then mapped to a formal target language. [22]

In [20], Malcolm Shroff and Robert B. France present how to formalize the class structures in UML using Z. The primary construct in Z is the schema which can be divided into two parts: a *declaration* part and a *predicate part* [21]. The focus of [20] is formalizing the classes and the relationships between classes such as *Association* and *Aggregation*.

[20] provides an example that formalizes a many-to-many relationship between companies and persons. The following class diagram shows the relationship.



We can define the "works for" association in Z schema as below.

$$
\begin{array}{|l}
\textit{Works\_for} \\
\hline
company : \mathbb{P}\, Company \\
person : \mathbb{P}\, Person \\
works\_for : Person \leftrightarrow Company \\
\hline
works\_for \in * \longleftrightarrow *[person, company] \\
\forall i, j : company \bullet (i.ident = j.ident \Leftrightarrow i = j) \\
\forall i, j : person \bullet (i.ident = j.ident \Leftrightarrow i = j) \\
\hline
\end{array}
$$

Also we can define the "employs" association in Z schema as below.

$$
\begin{array}{|l}
\textit{Employs} \\
\hline
employer : \mathbb{P}\, Company \\
employee : \mathbb{P}\, Person \\
employs : Company \leftrightarrow Person \\
\hline
\mathrm{dom}\, employs = employer \\
\mathrm{ran}\, employs = employee \\
\forall i, j : employer \bullet (i.ident = j.ident \Leftrightarrow i = j) \\
\forall i, j : employee \bullet (i.ident = j.ident \Leftrightarrow i = j) \\
\hline
\end{array}
$$

William McUmber and Betty Cheng introduce a general framework for formalizing a subset of UML diagrams based on a mapping between meta-models describing UML and a formal language [22]. In this framework, a UML formalization consists of a 4-tuple *(S; T; h; R)*. S and T are the meta-models of the source and target languages, respectively, and h is the homomorphism between S and T as described earlier. *R* is a set of formalization rules, which provide the specific mappings of the semi-formal source to a target formal specification language.

2.3.3 Formal Specification Languages

Domain-specific languages (DSLs) are languages tailored to a specific application domain. DSLs trade generality for expressiveness in a limited domain. [23]

Many DSLs have been introduced in different areas. For example, GeoScenario is an open DSL for autonomous driving scenario representation [24].

Object Constraint Language (OCL) is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Each OCL expression is written in the context of an instance of a specific type [25]. According to [26], OCL has the following shortcomings:

- OCL seems to be closer to an implementation language than to a conceptual language.
- OCL expressions are at times unnecessarily verbose.
- OCL expressions are often unnecessarily hard to read.
- OCL is not a stand-alone language - It always needs an accompanying UML class diagram.

Alloy is a language and a toolkit for exploring the kinds of structures that arise in many software designs. Here the structures include architectures, database schemas, network topologies, ontologies, and so on [28]. The starting point of Alloy is Z. The semantics of Alloy bridges the gap between Z and object models [27].

TLA+ is a high-level language for modeling programs and systems, especially concurrent and distributed ones. It comes with a model checker named TLC [29]. **PlusCal** is introduced as a pseudocode-like language to express the algorithm. PlusCal can be translated into TLA+ [30].

Rich data structures and expressiveness are the reasons that Amazon chooses TLA+ over Alloy to model its systems [31].

Tabular notations can be used to describe the events, conditions and transitions in a system. [32] shows how to use event tables, condition tables and mode transition tables to formally state the requirements. [33] presents an example of implementing formal methods, including tabular notations, at a nuclear power station.

2.3.4 Pros & Cons of formalization

Before using formal languages in requirements documentation, we would better understand the Pros and Cons. Here I list some pros & cons of using formal specifications.

Some Pros.

- Validation friendly. With formal specification, we may use tools like model checking or SAT/SMT solvers to validate that requirements.
- Simulation friendly. It is easier to simulate the behaviour specified in the formal specifications.
- Verification friendly. Formal requirements specs make it possible to do mathematical verification of design and implementation (code) as well as testing.
- Precision. Formal specs have the promise of being more precise and less ambiguous than natural language specs.

Some Cons.

- Time consuming, compared with requirements in natural languages.
- Steep learning curve. Formalized language requires different training and expertise compared with traditional requirement specification
- Expressiveness. We may miss the intent of the real requirement because of the limitation of math notations.
- Over-confidence. Many people do not realize we can make mistakes in mathematics.
- Difficulties in comprehension. The formal languages use math notations that engineers do not find very friendly or understandable. Subtle differences are often not appreciated or realized by readers or specifiers.

2.4 Requirements in a larger world

2.4.1 Modeling the operational environment

Most existing software development methodologies (object-oriented, structured or otherwise) have traditionally been inspired by programming concepts, not organizational ones, leading

to a semantic gap between the software system and its operational environment. [34]

To reduce the gap between the system and the operational environment, [34] proposes a development framework, named *Tropos*.

There are four phases in the *Tropos* framework:

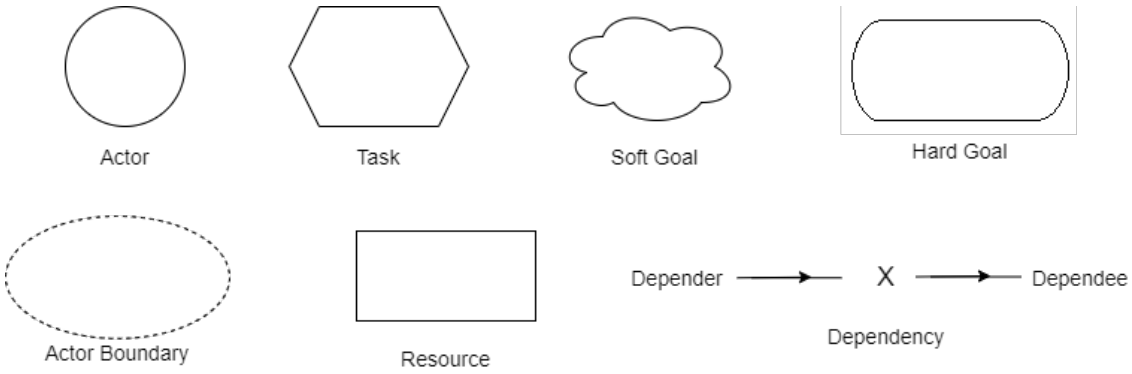
- Early requirements - A phase in which we need to understand the problem. The output of this phase is an organizational model which includes relevant actors.
- Late requirements - The system-to-be is described within its operational environment.
- Architectural design - System's global architecture is defined in terms of sub-systems, interconnected through data, control and other dependencies.
- Detailed design - the behavior of each component is defined in further detail.

The key concepts in *Tropos* include [35]:

- Actor - An actor models an entity that has strategic goals and intentionality within the system or the organization.
- Goal - A goal represents actors' strategic interests. We always distinguish hard goals and soft goals.
- Plan - A plan represents, at an abstract level, a way of doing something.
- Dependency - A dependency between two actors indicates that one actor depends, for some reason, on the other in order to attain some goals.
- Capability - The ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal.

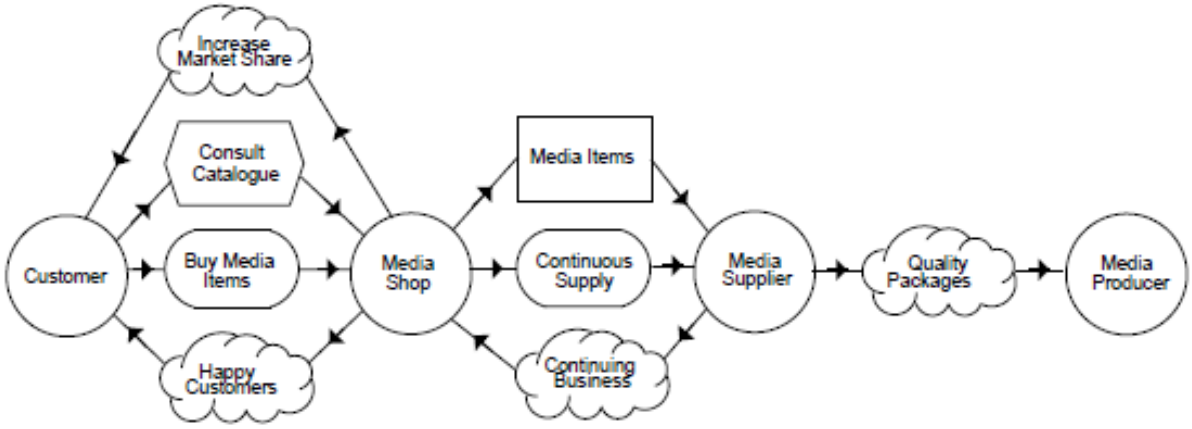
The foundation of *Tropos* is *i**, which is an agent-oriented modeling framework that can be used for requirements engineering, business process reengineering, organizational impact analysis, and software process modeling [36]. The *i** framework includes the *strategic dependency model* for describing the network of relationships among actors, as well as the *strategic rationale model* for describing and supporting the reasoning that each actor goes through concerning its relationships with other actors [36].

The major graphical symbols in i^* are shown in the following figure.



As an example, [34] uses i^* to create a model for a media shop. In this scenario, Media Shop is a store selling and shipping different kinds of media items such as books, newspapers, magazines, audio CDs, videotapes, etc. Media Shop customers (on-site or remote) can use a periodically updated catalogue describing available media items to specify their order.

The following figure shows the i^* model of the Media Shop.



2.4.2 Goal-based requirements analysis

A goal is an objective the system under consideration should achieve. [37]

Realizing the goals are not always well documented, Annie Anton proposes a method named **Goal-Based Requirements Analysis Method** (GBRAM) to identify, elaborate, refine and organize goals in [38]. GBRAM defines the following key concepts.

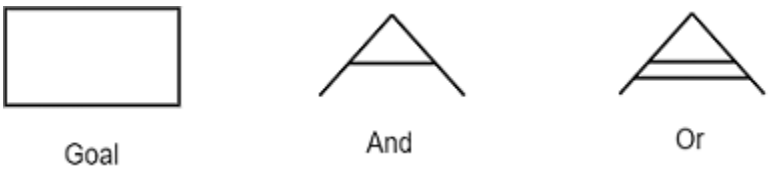
- Goals - High level objectives of the business, organization or system.

- Achievement goals - Objectives of some enterprise or system.
- Maintenance goals - Goals that are satisfied while their target condition remains true.
- Goal obstacles - Behaviours or other goals that prevent or block the achievement of a given goal.
- Agents - Entities or processes that seek to achieve goals.
- Scenarios - Behavioral descriptions of a system and its environment arising from restricted situations.
- Constraints - Requirements that must be met for goal completion.

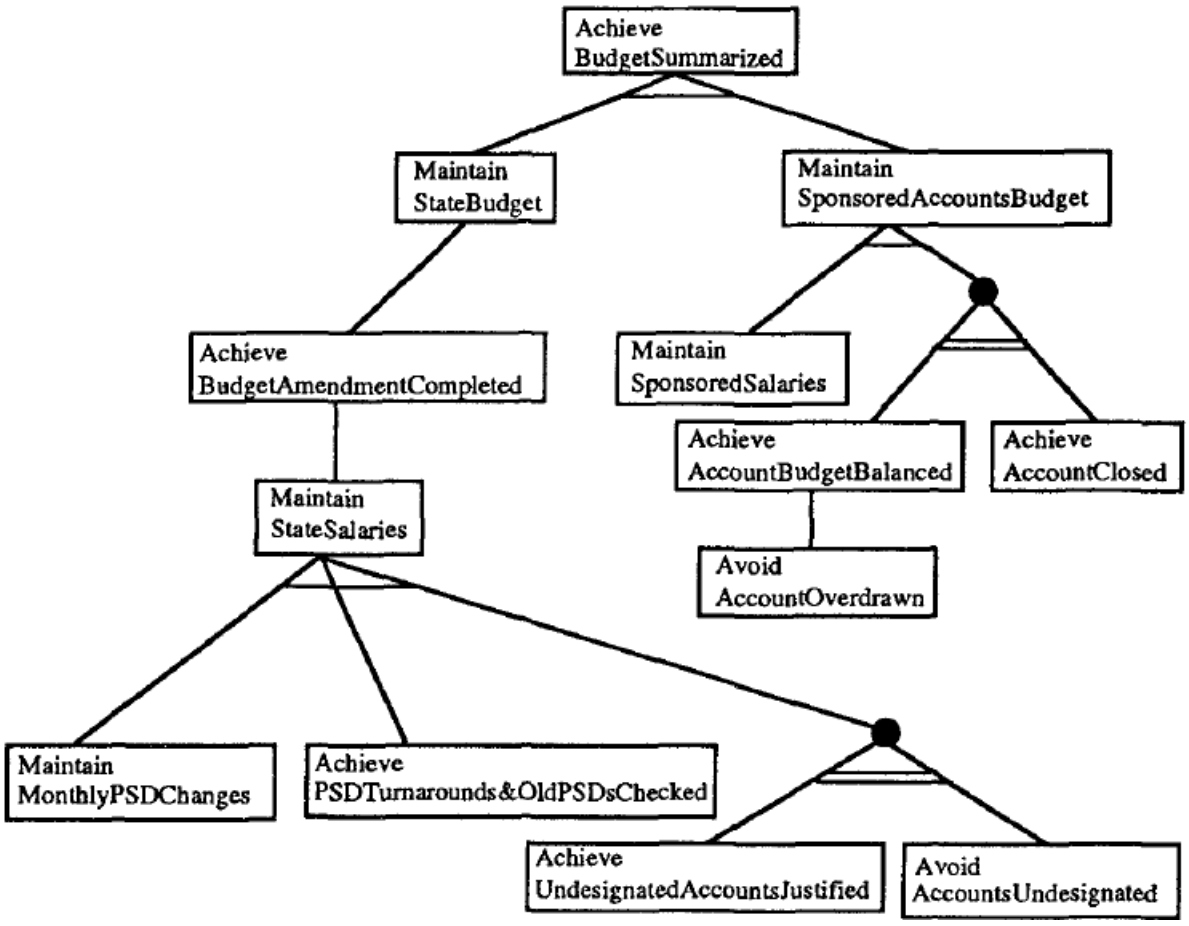
As an example, [38] uses GBRAM to analyze the goals for a Career Track Training System (CTTS), which is a part of a reengineering project for Air Force Base (AFB). Some goals in CTTS can be described in a table, as shown below.

Achievement Goals	Agents	Stakeholders	Goal Obstacles	Scenarios
Career portfolio created	employee	employee, supervisor	career portfolio not created	
Certification granted	AFB	AFB, employee	certification not granted	course taken doesn't qualify employee

In a complex system, there is always more than one goal. We can construct a goal hierarchy using the representation scheme presented in [39]. There are two kinds of relationships among sub-goals in the scheme: "And" and "Or". The basic graphical symbols in the scheme are shown in the following figure.



As an example of goal hierarchy, [40] shows a "Accounts Undesignated" scenario. This scenario explains how to maintain accounts of faculties in a school's salary management system. The following figure shows this scenario.



3 Future Work

There are still many issues in requirements documentation that need our work in the future.

In [41], Betty Cheng states some open questions in requirements engineering. These questions also need to be addressed in the requirements documentation phase, in my opinion.

- Security - There is no consensus on how security requirements themselves should be documented. Is security a nonfunctional requirement to be resolved and optimized at design time? Or should security requirements be realized as functional requirements?

- Product lines documentation - Feature models are commonly used to model a product-line core, but they quickly proliferate when used to model product-line instantiations.

- Self-management systems - These systems require different perspectives on what types

of requirements information should be documented, in contrast to traditional approaches, which typically focus on a static set of goals or functionality.

Another challenging problem lies in the area of formal specification languages. [31] presents two difficulties in applying formal languages: 1. Formal methods deal with models of systems, not the systems themselves. So the accuracy of models is of vital importance when we apply formal languages. 2. Formal methods require a large amount of training and effort to verify a relatively small code snippet. So I believe a lot of work is needed to provide mitigation to these difficulties. Furthermore, when we want to use formal languages to model a new feature on an existing system, we may need to model the existing system itself. In this case, an automatic or semi-automatic modelling approach will be very beneficial.

To fill the gap between academia and industry, I feel more tools are needed. In order to make useful tools, continuous cooperation among domain experts, developers and researchers is necessary.

References

[1] Kotonya, G & Sommerville, I (1998) *Requirements Engineering: Processes and Techniques.* John Wiley & Sons. ISBN 978-0-471-97208-2.

[2] Axel van Lamsweerde (2009) *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons. ISBN: 978-0-470-01270-3

[3] Mich, Luisa, & Franch, Mariangela. et. al. (2004) *Market Research for Requirements Analysis Using Linguistic Tools*

[4] ISO/IEC/IEEE Standard 29148 (2011) *Systems and software engineering — Life cycle processes — Requirements engineering*

[5] Ryan, Kevin. (1993) *The role of natural language in requirements engineering," in IEEE International Symposium on Requirements Engineering* pp. 240-242

[6] Zhao, Liping & Alhoshan, Waad & Ferrari, Alessio et. al. (2020) *Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study.* arxiv 2004.01099

[7] Dalpiaz, F. et al. (2018) *Natural language processing for requirements engineering: The best is yet to come*.doi: 10.1109/MS.2018.3571242.

[8] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J. (2013) *Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems*. pp. 3111–3119. arXiv:1310.4546

[9] Ferrari, A., Esuli, A. (2019) *An NLP approach for cross-domain ambiguity detection in requirements engineering*. Automated Software Engineering. June 2019.

[10] Gemkow, T., Conzelmann, M., et. al. (2018) *Automatic glossary term extraction from large-scale requirements specifications*. 26th IEEE International Requirements Engineering Conference

[11] Ferrari A, Spagnolo GO, Gnesi S. (2018) *Pure: A Dataset Of Public Requirements Documents.* doi:10.5281/ZENODO.1414116

[12] Daniel, Moody. (2009) *The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering*. IEEE Transaction on Software Engineering, VOL. 35, NO. 6

[13] DeMarco, T. (1978) *Structured Analysis and System Specification*. Yourdon Press.

[14] Jackson, M. (2001) *Problem Frames: Analyzing and Structuring Software Development Problems*. ACM Press/Addison-Wesley.

[15] Object Management Group (2017) *Unified Modeling Language specification*, version 2.5.1. https://www.omg.org/spec/UML/2.5.1/PDF

[16] Jackson, M. (2002) *Some Basic Tenets of Description*. Software & Systems Journal, Volume 1 Number 1, pages 5-9.

[17] Chen, P. (1976). *The Entity Relationship Model - Towards a Unified View of Data. ACM Transactions on Database Systems*, Vol. 1, No. 1, March 9-36

[18] Rumbaugh, J., Jacobson, I. and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison-Wesley Object Technology Series.

[19] Bruel, Jean-Michel & France, Robert. (1999). *Transforming UML Models to Formal Specifications.* Proceedings of the OOPSLA 98

[20] M. Shroff & R.B. France. (1997) *Towards a Formalization of UML Class Structures in Z.* Proceedings of the 21st Annual International Computer Software and Applications Conference (COMPSAC'97), pages 646–651. Los Alamitos, CA, USA, August 1997.

[21] Spivey, J. (1989) *The Z notation - a reference manual.* Prentice Hall International Series in Computer Science (1989).

[22] William McUmber & Betty Cheng (2001) *A general framework for formalizing UML with formal languages* Proceedings of the 23rd International Conference on Software Engineering

[23] M Mernik, J Heering, AM Sloane (2005) *When and how to develop domain-specific languages.* ACM computing surveys (CSUR) 37 (4), 316-344

[24] R. Queiroz, T. Berger and K. Czarnecki (2019) *GeoScenario: An Open DSL for Autonomous Driving Scenario Representation.* 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 2019, pp. 287-294, doi: 10.1109/IVS.2019.8814107.

[25] Object Management Group (2014) *Object Constraint Language Specification 2.4*

[26] Mandana Vaziri, Daniel Jackson (1999) *Some shortcomings of OCL, the object constraint language of UML.* TOOLS (34), 555-562

[27] Daniel Jackson (2001) *Alloy: a lightweight object modelling notation.* ACM Transactions on Software Engineering and Methodology (TOSEM) 11 (2), 256-290

[28] Daniel Jackson (2019) *Alloy: a language and tool for exploring software designs.* Communications of the ACM 62 (9), 66-76

[29] Yuan Yu, Panagiotis Manolios, Leslie Lamport (1999) *Model Checking TLA+ Specifications.* Advanced Research Working Conference on Correct Hardware Design and Verification Methods

[30] Leslie Lamport (2009) *The PlusCal Algorithm Language.* Theoretical Aspects of Computing-ICTAC 2009

[31] Chris Newcombe, Tim Rath, et. al. (2015) *How Amazon Web Services Uses Formal Methods.* Communications of the ACM, April 2015, Vol. 58, No. 4

[32] C. Heitmeyer, A. Bull, C. Gasarch, B. Labaw *SCR: a toolset for specifying and analyzing requirements.* COMPASS '95 Proceedings of the Tenth Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security' (1995): 109-122.

[33] Alan Wassyng, Mark Lawford (2003) *Lessons learned from a successful implementation of formal methods in an industrial project.* International Symposium of Formal Methods Europe, 133-153

[34] Castro, J., Kolp, M., Mylopoulos, J., (2002) *Towards Requirements-Driven Software Development Methodology: The Tropos Project* Information Systems, June 2002

[35] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J. *Tropos: An Agent-Oriented Software Development Methodology* Autonomous Agents and Multi-Agent Systems 8 (3), 203-236

[36] Eric Yu (1997) *Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering.* Proceedings of the 3rd International Symposium on Requirements Engineering (RE'97), Washington, USA, January 1997.

[37] Axel van Lamsweerde (2001) *Goal-Oriented Requirements Engineering: A Guided Tour.* 5th IEEE International Symposium on Requirements Engineering, Toronto, August 2001, 249-263

[38] Annie I. Anton (1996) *Goal-based requirements analysis.* Proceedings of the 2nd international conference on requirements engineering, 136-144

[39] Benner, K., Feather, M.S., Johnson, W.L., Zorman, L. (1992) *Utilizing Scenarios in the Software Development Process.* IFIP WG 8.1 Working Conference on Information Systems Development Process, 117–134

[40] A.I. Anton & W.M. McCracken & C. Potts (1994) *Goal Decomposition and Scenario Analysis in Business Process Reengineering.* International Conference on Advanced Information Systems Engineering, 94-104

[41] Cheng, Betty & Atlee, Joanne (2009) *Current and future research directions in requirements engineering.* Design Requirements Engineering: A Ten-Year Perspective, 11-43