Requirements Eng (2011) 16:75–99
DOI 10.1007/s00766-010-0111-y

ORIGINAL RESEARCH

A systematic review of transformation approaches between user requirements and analysis models

Tao Yue · Lionel C. Briand · Yvan Labiche

Received: 27 April 2009 / Accepted: 9 August 2010 / Published online: 26 August 2010

Abstract Model transformation is one of the basic principles of Model Driven Architecture. To build a software system, a sequence of transformations is performed, starting from requirements and ending with implementation. However, requirements are mostly in the form of text, but not a model that can be easily understood by computers; therefore, automated transformations from requirements to analysis models are not easy to achieve. The overall objective of this systematic review is to examine existing literature works that transform textual requirements into analysis models, highlight open issues, and provide suggestions on potential directions of future research. The systematic review led to the analysis of 20 primary studies (16 approaches) obtained after a carefully designed procedure for selecting papers published in journals and conferences from 1996 to 2008 and Software Engineering textbooks. A conceptual framework is designed to provide common concepts and terminology and to define a unified transformation process. This facilitates the comparison and evaluation of the reviewed papers.

T. Yue · L. C. Briand
Simula Research Laboratory, University of Oslo,
P.O. Box 134, Lysaker, Norway

L. C. Briand
e-mail: briand@simula.no

T. Yue (✉) · Y. Labiche
Software Quality Engineering Lab, Carleton University,
1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada
e-mail: tao@simula.no

Y. Labiche
e-mail: labiche@sce.carleton.ca

Keywords Systematic review · Requirements · Analysis model · Transformation · Traceability · Natural language

1 Problem definition

One of the basic principles of Model Driven Architecture (MDA) [29] is model transformation. To build a software system, a series of transformations is performed: transformation from requirements to Platform Independent Model (PIM) (the analysis model), transformation from PIM to Platform Specific Model (PSM) (the design model), and transformation from PSM to code. However, the transformation from requirements to an analysis model is not part of the MDA lifecycle, which starts from an analysis model and ends with deployed code [29]. The reason of this exclusion is perhaps that requirements are mostly in a textual form, which is not a model formal enough to be understood by computers. As a result, requirements are not suitable for automated transformations, and only manual heuristics, such as Abbott's heuristics [4], are in general followed [9, 32]. However, if a (semi-) automated transformation technology from requirements to an analysis model were devised, it would help fill an important gap in the MDA software development lifecycle.

If these transformations can be automated, full traceability from requirements (through PIM and PSM) to the ultimate software code can be obtained at the same time. Traceability is the ability to link requirements to corresponding analysis and design models, code, test cases, and other software artifacts. Traceability is important during software development because it enables engineers to understand the connections between various artifacts of a software system, and it is also used to determine whether

developers have refined requirements into high-level design components, then lower-level design components, and eventually built them into executable code. Traceability is also mandated by numerous standards (e.g., IEEE Std. 830-1998 [3]).

Though the importance of traceability has been well recognized, traceability is still not widespread in practice because of the reasons identified in [43], including: different languages used in different software development phases and insufficient tool support for traceability creation and maintenance which is critical in practice since models tend to be large, and we can therefore expect to have to create and maintain large numbers of traceability links. MDA provides new opportunities for establishing traceability links through transformations. With the support of MDA strategies, transformation-based techniques generate traceability links along with the generation of the artifacts (e.g., analysis models), which may be represented in different languages and at different levels of abstractions. However, the integration of transformations with traceability is still not well developed, and more research on automated traceability creation is needed [5]. One of the promising methods conforming to MDA is the utilization of use-case-centric methods (e.g., IBM Rational Unified Process [30]). As explained in [5], a use-case-centric method requires that software requirements be specified as use cases (requirements), which are further transformed into use case realizations (analysis model). These use case realizations are then transformed into low-level artifacts (code). Traceability links can be easily created when these transformations are performed.

The IBM Rational Software Architect (RSA) [22] is a tool that has taken an initial step to realize a use-case-centric method. IBM RSA currently supports transformation from a use case template to an analysis model template, though it is currently coarse-grained. For example, actors are transformed into boundary classes; each use case is transformed into a boundary class, a controller class, and a use case realization with a default, empty *Interaction* (a Unified Modeling Language (UML) [38] model element). The textual description of each use case is not analyzed and transformed though; therefore, fine-grained transformation is not supported.

This paper reports on a systematic review that focuses on approaches for transforming requirements into analysis models. The intent is to determine whether these approaches have the capability to transform requirements into an analysis model and at the same time establish traceability links between them, and how automated, efficient, and complete the transformation process is. This review systematically selected, investigated, and compared 20 primary studies (16 approaches) for transforming requirements into an analysis model. In order to facilitate the synthesis and comparison of these approaches, we designed a conceptual framework providing common concepts, terminology, and a unified transformation process for the comparison and evaluation of transformation approaches. A set of evaluation criteria, derived from the conceptual framework, is proposed to evaluate approach in a precise and structured manner. These criteria can be adapted to evaluate future research works on the same topic.

We observed from the systematic review results that existing approach cannot easily and realistically be applied on real systems for documenting requirements or that they are not able to (semi-) automatically generate a complete, consistent analysis model, which is expected to model both the structure and behavior of the system at a logical level of abstraction. Based on a careful analysis and evaluation of each aspect of the reviewed approaches, we identify open issues and make recommendations for future work. We also conclude that future promising approaches will likely match the following pattern: automatically and efficiently transform a use case model using reasonable restrictions to natural language, with or without domain-specific information provided in a glossary, into a complete, correct and consistent UML model comprising both structural and behavioral aspects using one intermediate model and fully automatable requirements pre-processing techniques.

The scope of the systematic review is further defined in Sect. 1.1, followed by a description of the structure of the paper (Sect. 1.2).

1.1 Scope

This paper reports on a systematic review of approaches for transformation between requirements and analysis models. In this section, we refine the scope of the systematic review by defining, more precisely, the relevant, fundamental concepts.

A requirement is defined in [55] as "a statement that identifies a necessary attribute, capability, characteristic, or quality of a system in order for it to have value and utility to a user". Requirements should be easy to understand since they are usually written as a means for communication between different stakeholders (e.g., users, developers). There are many different ways to document requirements. One common way is to use textual descriptions only. Other ways to document requirements include use cases and customized document templates. For some systems (e.g., safety critical systems), requirements may even be documented as formal specifications. In our systematic review, we limit our scope to requirements documented using pure textual descriptions, use cases, or customized document templates. We exclude formal methods (mathematical descriptions) from our systematic review since they are less often used to formalize requirements in practice and present very different problems

than textual requirements (i.e., mapping between formal languages).

An analysis model is a description of what a system is required to do functionally and aims to be less ambiguous and more correct and consistent than textual requirements [9]. In a typical object-oriented software development process, the analysis model is usually derived from requirements. It is typically represented as a UML model containing various diagrams and possibly constraints. Our systematic review is, however, not limited to UML models. Other representations are also taken into account, as they often share similar object-oriented concepts. Other well-known notations include, for example, message sequence charts (MSC) or entity relationship models (ERM).

1.2 Structure

The objective of this systematic review, its research questions, search strategy, and data extraction, synthesis, and comparison strategy are described in Sect. 2. The conceptual framework being used to synthesize and compare the reviewed papers and to derive comparison criteria is described in Sect. 3. The evaluation criteria used to evaluate related works is presented in Sect. 4. Section 5 presents the detailed analysis and comparison of the selected related works. The open issues and suggestions are discussed in Sect. 6, followed by the conclusion given in Sect. 7.

2 Systematic review method

A systematic review follows a well-defined method to identify, analyze, synthesize, evaluate, and compare all available literature works relevant to a specific research topic [28]. A systematic review is an important piece of work since it summarizes existing techniques concerning a research interest, it identifies further research directions, and it provides a framework to position new research activities [28]. Several discrete activities (i.e., guidelines) are recommended for all systematic reviews in software engineering [28], among which are four recommended essential steps that we adopted: first, we clearly define research questions that the systematic review is expecting to answer (Sect. 2.1); second, we develop a search strategy (Sect. 2.2), which includes a paper selection procedure, resources to be searched, and inclusion and exclusion criteria; third, we use the search strategy to identify the relevant research works; Last, we analyze, synthesize, and compare the related works to answer the research questions.

The main objective of this systematic review is to develop a conceptual framework to synthesize and compare the related works proposing approaches of transformation between requirements and analysis models. Based on the synthesis and comparison results, we summarize and analyze the reviewed works, identify open issues, and provide suggestions for future research.

2.1 Research questions

In order to examine the evidence of transforming requirements into analysis models using different transformation approaches, this systematic review aims to answer the following research questions: (1) What are the different approaches used for transforming requirements into analysis models? (2) What are the current limitations of these approaches? (3) What are the open issues to be further investigated?

The first research question is further divided into the following sub-questions: (i) What are the different requirement representations (e.g., use cases, pure textual specifications, and customer-specified templates) required by these approaches? Is it difficult for users to document such requirements? Is there any tool support? (ii) What kinds of analysis models (e.g., UML diagrams and message sequence charts) can be generated by these approaches? Does a generated analysis model contain both the structural and behavioral aspects of a system? (iii) Are there any intermediate models used during transformation from requirements to analysis models? How do they affect the efficiency of the transformation? (iv) Are these approaches automated, automatable, semi-automated, or manual? Are there algorithms presented in the approaches? (v) What steps are taken by each approach to transform user requirements into analysis models? (vi) Do approaches include traceability management support? (vii) Have any case studies been performed to evaluate the approaches? If yes, what results have been obtained? What other evaluation methods (besides case studies) have been applied to evaluate these approaches?

2.2 Search strategy

In this section, we present the search strategy that we applied to select papers to be reviewed. We initiated our search by identifying a query string being used to perform electronic searches, based on our research questions (Sect. 2.2.1). Then, we searched five electronic databases using this query string (Sect. 2.2.2). In addition, as a complement to the electronic search, we performed manual search in specific journals and conference proceedings and also manually checked Software Engineering textbooks (Sect. 2.2.2). We then scanned all the sources resulting from this two-stage search to select the works to be included in the review. During this step, we applied inclusion/exclusion criteria (Sect. 2.2.3) to select primary studies. For each paper, we read the paper's title and abstract to see whether

it was relevant to our research topic. If the title and abstract of the paper could not help us make a decision, we further checked the paper's full text. In order to augment our collection of primary studies, we scanned the reference lists of all the identified primary studies to identify additional papers. Furthermore, we also went through publication lists of primary studies' authors to make sure that the most recent publications on the same or similar topics were included. The statistic data of included primary studies are presented in Sect. 2.2.4.

2.2.1 Identification of query string

Based on the research questions (Sect. 2.1), we identified three groups of search terms: population terms, intervention terms, and outcome terms. The population terms are the keywords that represent the domain of transforming requirements into analysis models, such as requirements analysis, requirements refinement, use cases realization, and domain modeling. The intervention terms are the keywords that represent the techniques applied in the population to achieve an objective. In our case, the techniques for transforming requirements into analysis models could be very different; therefore, we decided to use general terms like transformation, generation, and linguistic analysis. The outcome terms represent different types of analysis models, which could be generated.

To form the query string, we used a disjunction of the keywords of each term group and then used the conjunction of the three groups of terms. The following three groups of terms were used to form the query string:

Population terms: requirements analysis; requirements engineering; requirements refinement; requirements formalization; use cases analysis; use cases formalization; use cases realization; object oriented analysis; object-oriented analysis; object identification; domain modeling.

Intervention terms: automated transformation; automatic transformation; transformation; transform; transforming; translation; translate; translating; derive; deriving; generation; generate; generating; linguistic analysis; linguistic analyze; natural language processing.

Outcome terms: analysis model(s); object model(s); static model(s); dynamic model(s); UML model(s); class diagram(s); sequence diagrams(s); interaction diagram(s); activity diagrams(s); state machine(s); statechart(s); class model(s); interaction model(s); object oriented model(s); object-oriented model(s); object(s); class(es); message sequence chart(s).

2.2.2 Electronic and manual search

We performed electronic search within five electronic databases: IEEE Xplore, ACM Digital Library, Compendex, Inspec, and SpringerLink, using the query string we described earlier. Each time, we modified the query string to fit the format requirements of the electronic database before applying it. We also manually searched all published papers from 1996 to 2008 in nine potentially relevant, peer-reviewed journals: IEEE Transactions on Software Engineering, Automated Software Engineering, Requirements Engineering Journal, Journal of Natural Language Engineering, ACM Transactions on Software Engineering and Methodology, Journal of Systems and Software, Software and Systems Modeling, Information and Software Technology, and Data & Knowledge Engineering. We also manually searched all published papers from 1996 to 2008 in five potentially related conference proceedings: ACM/IEEE International Conference on Software Engineering, IEEE International Conference on Software Maintenance, IEEE/ACM International Conference on Automated Software Engineering, IEEE International Conference on Model Driven Engineering Languages and Systems and the former UML workshops, and IEEE International Requirements Engineering Conference. We also manually searched Software Engineering textbooks (e.g., [9, 32]) that describe transformations from requirements to analysis models.

2.2.3 Inclusion/exclusion criteria

Approaches for transforming requirements into analysis models vary a great deal as different requirement representations are adopted as inputs, and/or different analysis models are generated. Therefore, it is absolutely necessary that we define thorough inclusion/exclusion criteria to select the primary studies that can answer our research questions (Sect. 2.1) and also conform to our research scope (Sect. 1.1). We used the following inclusion/exclusion criteria:

- Papers irrelevant to the transformation of requirements are excluded. For example, the papers discussing information retrieval techniques that are used to recover traceability links (one of the capabilities of transformation approaches) between software artifacts are excluded.
- Papers proposing transformation approaches between software artifacts that are out of our review scope are excluded. For example, transformations between requirements given as formal specifications and design models or code are excluded (e.g., [53]).
- When encountering more than one paper describing the same or similar approaches, which were published in different venues, we only included the most recent one or the one with the most complete description of the approach.

- When a single approach is presented in more than one paper describing different parts of the approach, we included all these papers, but still considered them as a single approach.
- Papers with insufficient technical information regarding their approaches were excluded. For example, the papers that do not provide a detailed description on requirements representations, intermediate models (if any), and transformation techniques, are considered incomplete and are excluded (e.g., [11]).
- Software Engineering textbooks (e.g., [9, 31–33, 42, 50]) share similarities with respect to the requirements to analysis model transformation. They all describe an intuitive set of heuristics to guide designers identify objects, attributes, and associations from a requirement specification, mapping parts of speech (e.g., nouns, verbs and adjectives) to model elements (e.g., objects, operations, and attributes). Some of these books refer to Abbott's heuristics [4] (e.g., [9, 32, 42, 50]), others provide similar heuristics to Abbott's (e.g., [33]), and a third group extends Abbott's heuristics (e.g., [9, 31]). These textbooks suggest that these heuristics be applied manually (no transformation approach is described) and no requirements pre-processing technique is applied. According to our taxonomy of approaches (Sect. 5), all those approaches fall into the same category. Since Abbott's heuristics is the primary study that is mostly used or referenced, we therefore only include Abbott's heuristics [4] as one of the primary studies. We only mention individual textbooks when we discuss their transformation rules (heuristics) that do not refer to Abbott's or extend Abbott's.

2.2.4 *Statistics from included primary studies*

The electronic search results are summarized in Table 1. A total of 451 papers were found. After eliminating duplicates, 361 papers remained to be further investigated.

After filtering the results (361 papers) of the electronic search by applying the inclusion/exclusion criteria, we identified 11 papers [6, 14, 19, 21, 24, 25, 36, 40, 48, 51, 54] to include. The manual search of journals and conference proceedings yielded an additional six primary studies (i.e., [10, 17, 18, 20, 47, 49]), and two of them (i.e., [18, 49]) are more recent discussions and therefore replaced two of the 11 papers identified from the electronic search (i.e., [19, 48]). We scanned the references and the authors' publication lists of all the 15 primary studies (11 + 6−2). Five new papers were identified (i.e., [34, 35, 44, 45, 52]), and one of them (i.e., [52]) (with a more complete description of the approach) replaced one of the already identified 15 primary studies (i.e., [51]). Among all these 19 primary studies (15 + 5−1), three groups of papers (i.e., [34, 52], [18, 35, 44], and [45, 49]) describe three individual approaches. The papers in each group together describe a single approach. Therefore, eventually a total of 15 approaches (19 primary studies) were included in the review (i.e., [6, 10, 14, 17, 18, 20, 21, 24, 25, 34–36, 40, 44, 45, 47, 49, 52, 54]). In addition to these 15 approaches (19 primary studies), we also included Abbott's heuristics [4] as one of the primary studies, as discussed in Sect. 2.2.3. In total, we included 16 approaches (20 primary studies).

Table 1 Summary of electronic search results

Electronic databases	Query results	After removing duplicates
IEEE Xplore	179	179
ACM Digital Library	17	15
Compendex	86	66
Inspec	83	34
SpringerLink	86	67
Total	451	361

3 Conceptual framework

We designed a conceptual framework to extract and synthesize data from the primary studies in a systematic and precise way. The conceptual framework is composed of a static model describing common concepts and their relationships (Sect. 3.1), five taxonomies classifying and specifying existing work according to five aspects, specifically the kinds of requirements, the rules imposed on requirements, the types of analysis models, requirements pre-processing approaches, and requirements transformation approaches (Sect. 3.2), and a general transformation process model (Sect. 3.3). This framework defines the common concepts and terminology needed for analysis, synthesis, and comparison of the primary studies. This is paramount since, for instance, different approaches may apply the same techniques but refer to them using different names. The framework therefore provides a way to unify the description of related works. The comparison and evaluation criteria are derived from this framework (Sect. 4).

3.1 Static model

In this section, we formalize the notions of transformation, traceability link, requirement, and analysis model by means of a metamodel. The metamodel is presented using the class diagram in Fig. 1. It illustrates the main concepts of our review framework and their relationships.

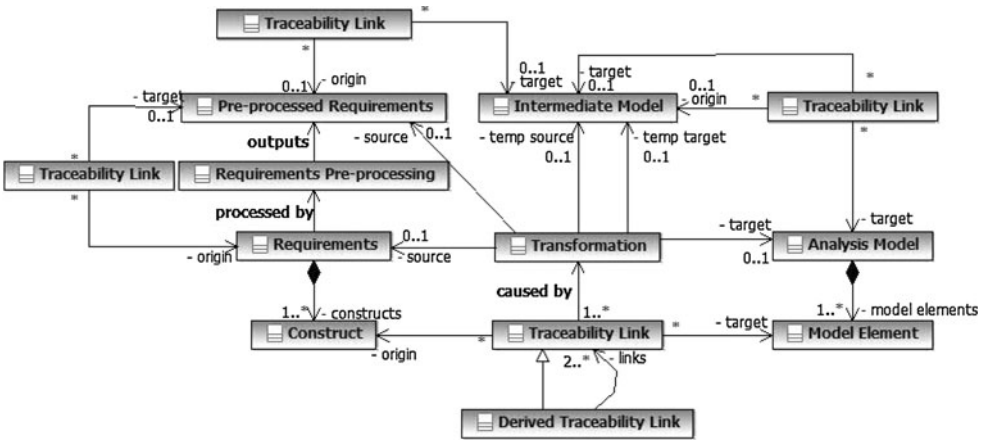

Fig. 1 Static model (class `Traceability Link` appears four times for layout purposes)

As shown in Fig. 1, `Requirements` can be transformed into an `Analysis model` either directly or indirectly. For direct transformation, there is no `Intermediate Model`, so only one `Transformation` is required: its `source` is the requirements and its `target` is the analysis model. For indirect transformation, one or more `Intermediate Models`[1] are used to bridge the gap between the requirements and the analysis model. Intermediate models function as a temporary `source` or `target` of the transformations, which are either from the requirements (`source`) to the first intermediate models (`temp target`), between two different intermediate models (one is `temp source` and the other is `temp target`), or from the last intermediate model (`temp source`) to the analysis model (`target`).

`Requirements` are composed of one or more `Constructs`,[2] while an `Analysis Model` is composed of one or more `Model Elements`.[3] Instances of `Traceability Link` are established between the constructs of the requirements and the model elements of the analysis model. For transformation-based traceability establishment, creating a traceability link is caused by a transformation. When traceability links can be established between a series of models (in the case of intermediate model(s)), we must derive traceability links between the constructs of the requirements and the model elements of the analysis model, these links being modeled as instances of class `Derived Traceability Link`.

Since requirements are textual specifications, they usually need to be pre-processed either manually or automatically before they are inputted to the transformation. One or more `Requirement Pre-processing` steps may be taken to transform `Requirements` into a `Pre-processed Requirements`, which is further transformed into either an analysis model or an intermediate model if one exists. During a series of transformations, traceability links should be established for the source and target of each transformation, for example, between the requirements and the pre-processed requirements, between the pre-processed requirements and the first intermediate model, between intermediate models if more than one exists, and/or between the last intermediate model and the analysis model.

3.2 Taxonomies

In this section, we define taxonomies to classify and specify the important techniques and terminology used in the primary studies. In the later sections of this paper, these taxonomies will be referred to in multiple places.

The taxonomy of requirements (Sect. 3.2.1) classifies different requirements representations, domain-specific information, and whether restricted natural language (NL) is applied. A restricted NL is a subset of a natural language, used to restrict its grammar and vocabulary, mostly for the purpose of reducing or eliminating ambiguity and complexity in its usage. The taxonomy of restriction rules (Sect. 3.2.1.1) classifies different types of restriction rules used for requirements written in restricted NL. The taxonomy of analysis models (Sect. 3.2.1.2) unifies different analysis models. We also provide a taxonomy of requirement pre-processing approaches in Sect. 3.2.1.3 to distinguish them at a certain level of abstraction. Last, a taxonomy of approaches for (pre-processed) requirements transformation is presented in Sect. 3.2.5.

3.2.1 Taxonomy of requirements

In order to generate analysis models from requirements and further establish traceability links between them, it is

[1] If multiple intermediate models are used, they are ordered.

[2] A construct can be a sentence, or an actor, if requirements are presented as use cases, for example. We do not distinguish different constructs in this conceptual static model.

[3] If the analysis model is presented as a UML model, then model elements are UML model elements. Other representations can be used equally.

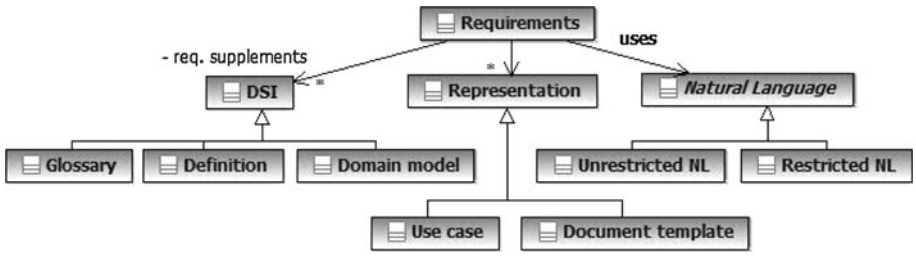

Fig. 2 Taxonomy of requirements

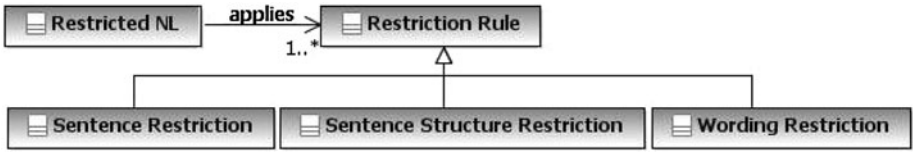

Fig. 3 Taxonomy of restriction rules

important to understand requirements from the following three aspects: which kinds of requirements supplements[4] are required (sub-taxonomy Domain Specification Information: `DSI` in Fig. 2), how requirements are represented (sub-taxonomy `Representation` in Fig. 2), and whether restricted NL is used when specifying requirements (sub-taxonomy `Natural Language` in Fig. 2). These sub-taxonomies, discussed in the following sub-sections, therefore represent what kinds of requirements are encountered in the literature.

3.2.1.1 Requirements representation In many situations, requirements are represented as `Use cases`. It is also possible that a customized `Document template` is applied to document requirements. Requirements can also be represented using more than one such representation. If no representation is used, then requirements are simply expressed in unstructured natural language. A use case is "the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system. [55]" A use case represents an interaction between a primary actor and other actors, and the system. This interaction is presented as sequences of simple steps (also called flow of events). Use cases are documented following a use case template. There is no standard template, and users typically choose the template that works for them, or is required by a project or a CASE tool. Some companies and organizations rather apply their own `Document Templates` for requirements documentation. These document templates are customized for special purposes such as facilitating requirements elicitation.

3.2.1.2 Requirements supplements (DSI) Domain-specific information is a necessary input for some of the approaches used to transform requirements into analysis models. It is either captured using a `Glossary`, `Definition`, and/or `Domain model`. A `Glossary` describes and classifies all the domain-specific terms used in requirements. A `Definition` [6] defines the notational short hand for expressing requirements in a succinct, practical, and domain-specific way. A `Domain model` is created to document the key concepts and the vocabulary of an application domain. It describes the various concepts involved in the application domain and their relationships. It is usually represented as a class diagram with possibly constraints in the object constraint language (OCL) [37].

3.2.1.3 Natural language (NL) Requirements can be written using either an `Unrestricted NL` or a `Restricted NL`. A restricted NL is also called a controlled NL. It is a subset of natural language obtained by restricting the grammar and vocabulary. It aims to reduce ambiguity, redundancy, size and complexity of requirements, and to facilitate automated analysis.

3.2.2 Taxonomy of restriction rules

As shown in Fig. 3, we classify the `Restriction Rules` of the `Restricted NL`s used in the literature into three types: `Sentence Restriction`, `Sentence Structure Restriction`, and `Wording Restriction`. A sentence is a group of words that are put together to mean something [2], and it is expected to have a subject and a verb. For example, the restrictions on allowed choices of tenses of a verb, on choices of singular or plural forms of a noun, are thought of as sentence restrictions. "Use active voice rather than passive voice" is another example of such a restriction. Sentence structure restrictions put restrictions on the structure of a compound sentence. A compound sentence has many clauses. These clauses are joined together with conjunctions, punctuation, or both [2]. For example, "only if–then structure is allowed to describe conditional sentence" is

[4] Requirements supplements refer to documents that clarify the terminology used in requirements.

a restriction on sentence structure. Wording restrictions restrict the choice of words and the way in which they are used: e.g., "use only keywords be or become to express a generalization relationship between the subject and the object of a sentence".

3.2.3 Taxonomy of analysis models

An analysis model is typically presented as a UML model, but not necessarily limited to it. A complete analysis model should describe two aspects of a system: `Structure` and `Behavior`. The structure (or static) aspect emphasizes the static structure of the system using classes, objects, attributes, operations, relationships, etc., while the behavior (or dynamic) aspect emphasizes the dynamic behavior of the system by showing interactions among objects, internal state changes, etc. As shown in Fig. 4, we classify the different presentations of the `Structure` aspect used in the literature into four types: `Class Diagram`, `Object Diagram`, Entity Relationship Model (`ERM`), and `Architecture Concept`. As two types of UML diagrams, class diagrams are composed of classes, attributes, operations, and relationships among the classes, and object diagrams describe objects and links. `ERM` was proposed in the early 70s to document the concepts of entity, relationship, types, and roles. `Architecture Concept` is used in [20] to present the concepts of components, connectors, and architectural patterns (e.g., client–server). The `Behavior` aspect is classified into five types: `Sequence Diagram`, `State Machine Diagram`, `Activity Diagram`, Data Flow Graph (`DFG`), and Message Sequence Chart (`MSC`). Sequence, state machine, and activity diagrams are three commonly used UML diagrams for describing the behavior of a system from three different views; sequence diagrams describe object interactions as messages, activity diagrams show the overall flow of control, and state machine diagrams describe state-based behavior. Message sequence charts are very similar to sequence diagrams. Data flow graphs represent data dependencies between operations.

3.2.4 Taxonomy of requirements pre-processing approaches

Most requirements are textual and have to be pre-processed (using NL processing techniques) before being used as the input for the next step's transformation. There are usually five types of pre-processing techniques (Fig. 5) that can be used in isolation or combined: `Lexical Analysis`, `Syntactic Analysis`, `Semantic Analysis`, `Categorization`, and `Pragmatic Analysis`.

`Lexical Analysis`, also called token generation, is the process of converting a sequence of characters into a sequence of tokens [55]. It is composed of the following processing steps: tokenization, sentence splitting, part-of-speech (POS) tagging, and morphological analysis. *Tokenization* is used to separate words and punctuation, and identify numbers. *Sentence splitting* identifies sentence boundaries within a given text. *POS tagging* identifies words as nouns, verbs, adjectives, etc. *Morphological analysis* returns the root and suffix of each word. `Syntactic analysis`, also called syntactic parsing, is the process of analyzing a sequence of tokens to determine grammatical structure with respect to a given formal grammar [1]. The output is usually a syntactic parse tree. `Semantic analysis` is the process of adding semantic information to a parse tree [1], typically by using domain-specific information (i.e., `DSI`). `Categorization` is the process of recognizing, differentiating, and classifying requirements for some specific purpose and is usually performed manually. `Pragmatic analysis` eliminates ambiguities and inconsistencies in requirements. For instance, pragmatic analysis can be used to check the consistency of a new piece of information before it is actually added to existing requirements [36].

3.2.5 Taxonomy of transformation approaches

Pre-processed requirements are further transformed into an analysis model or an intermediate model. Three types of transformations can be identified: `Rule based`, (the most commonly used in the literature) `Ontology based`, and `Identity Transformation` (Fig. 6). `Rule based`

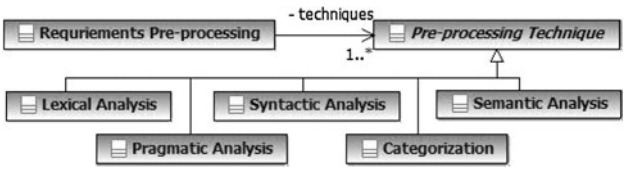

Fig. 5 Taxonomy of requirements pre-processing approaches

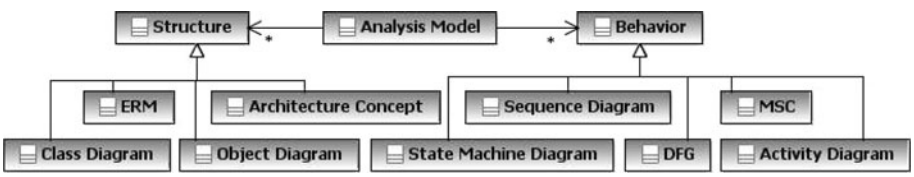

Fig. 4 Taxonomy of analysis models

transformation utilizes a set of predefined `Transformation Rules`. An ontology is a "shared vocabularies for describing the relevant notions of a certain application area, whose semantics is specified in a (reasonably) unambiguous and machine-processable form" [8]. An ontology model is built when NL sentences are processed. This ontology model acts as an intermediate model that is further transformed into an analysis model. Such transformations are called `Ontology-based` transformations. An `Identity Transformation` transforms a model (source) into another model (target) without change in information content: the two models describe the same concepts but with different representations. A `Pattern-based` transformation transforms source patterns into target patterns. A source pattern describes and organizes a set of source elements; while a target pattern describes and organizes a set of target elements.

3.3 Process model

We use an activity diagram (Fig. 7) to model the overall process of transforming requirements into an analysis model. First, requirements are pre-processed by applying one or more pre-processing techniques (Sect. 3.2.1.3), resulting into pre-processed requirements (step 1). If there is no intermediate model, then the pre-processed requirements are transformed directly into an analysis model (step 6); otherwise, the pre-processed requirements are transformed into an intermediate model (step 2). If there is more than one intermediate model involved, then transformations between these intermediate models are performed (step 3). Step 3 can be performed more than once, depending on the number of intermediate models. For example, if there are three intermediate models, step 3 is performed twice. Then step 4 transforms the last interme-

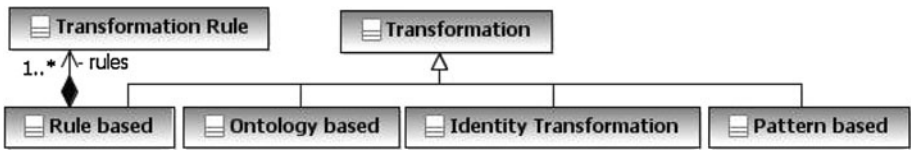

Fig. 6 Taxonomy of transformation approaches

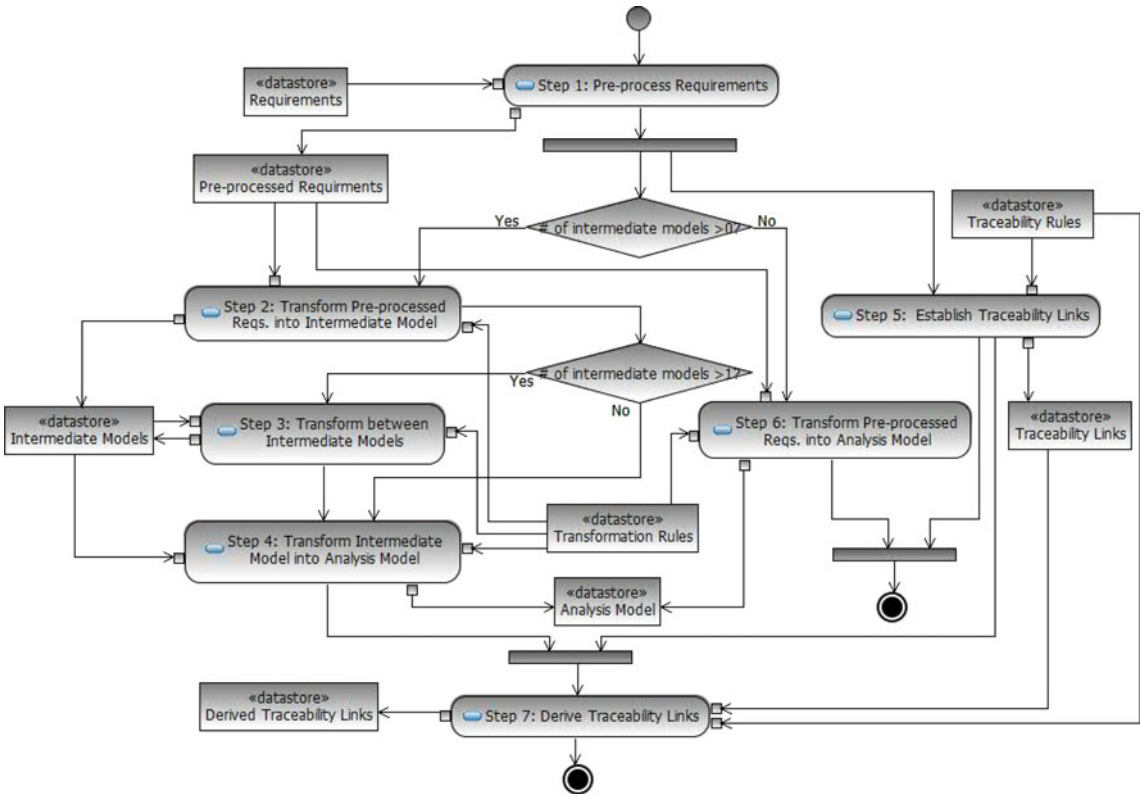

Fig. 7 A generic transformation process

diate model into the analysis model. While steps 2, 3, and 4 (or 6 if no intermediate model is used) are performed, traceability links are established between the source model and the target model of transformations (step 5). The output of this step is several sets of traceability links either between the requirements and the first intermediate model, between two intermediate models, or between the last intermediate model and the analysis model. Finally, we derive traceability links for the requirements and the generated analysis model (from step 4) from the sets of traceability links involving intermediate models (step 7). If there is no intermediate model (i.e., step 6 is taken), step 7 is obviously not needed. The output of the whole process is an analysis model, and a set of traceability links between the requirements and the generated analysis model.

4 Evaluation criteria

Our evaluation criteria are derived from the conceptual framework discussed in Sect. 3. Before specifying them, we first clarify their mapping to the conceptual framework (Fig. 8). As we have discussed in Sect. 3, the conceptual framework is composed of a `Static model`, five `Taxonomies` and one `Process model`. The evaluation criteria, being discussed in this section, are used to evaluate each reviewed approach in terms of their inputs (i.e., `Difficulty of documenting requirements`), their outputs (i.e., `Completeness of analysis models`), and their transformation approach from the following six aspects: `Automation`, `Efficiency`, `Evaluation`, `Traceability capability`, `Structuredness of transformation rules`, and `Completeness of transformation rules`. As shown in Fig. 8, for example, the evaluation criterion `Difficulty of documenting requirements` (Sect. 4.1) is derived from the `Taxonomy of requirements` (Sect. 3.2.1). Notice that the `Static model` of the conceptual framework (Fig. 1) formalizes a number of basic notions such as transformations and requirements. The static model is not directly related to any evaluation criteria; however, the taxonomies and the process model are all dependent on it. Last, note that the criterion `Evaluation methods in primary studies` reports, for example, on the number and size of the case studies performed and it is not therefore traced back to the conceptual framework.

The conceptual framework is used to extract and synthesize data from the primary studies in a systematic and precise way and then these data, presented as a table (Table 2), is analyzed according to the evaluation criteria, leading to the evaluation results reported in Sect. 5, in which we also summarize the restriction and transformation rules used in the approaches. As shown in Fig. 8, these two summaries are traced back to the taxonomy of restriction rules and the taxonomy of transformation approaches, respectively.

4.1 Evaluation criterion for requirements

We need to assess how difficult it is to document requirements in the format required by a specific approach. We do so by considering whether any `DSI` (i.e., `Glossary`, `Definition`, and `Domain Model`) is required, whether a restricted NL is enforced to write requirements, and whether the requirements representation is commonly used and well supported in practice. If an approach requires `DSI`, a restricted NL is enforced, and the requirements are represented using a specific template (e.g., not standard or commonly used), documenting requirements is deemed difficult. At the other end of the spectrum, if an approach does not require `DSI`, applies unrestricted NL, and applies commonly used requirements representations (e.g., use cases descriptions), then requirements are deemed easy to document.

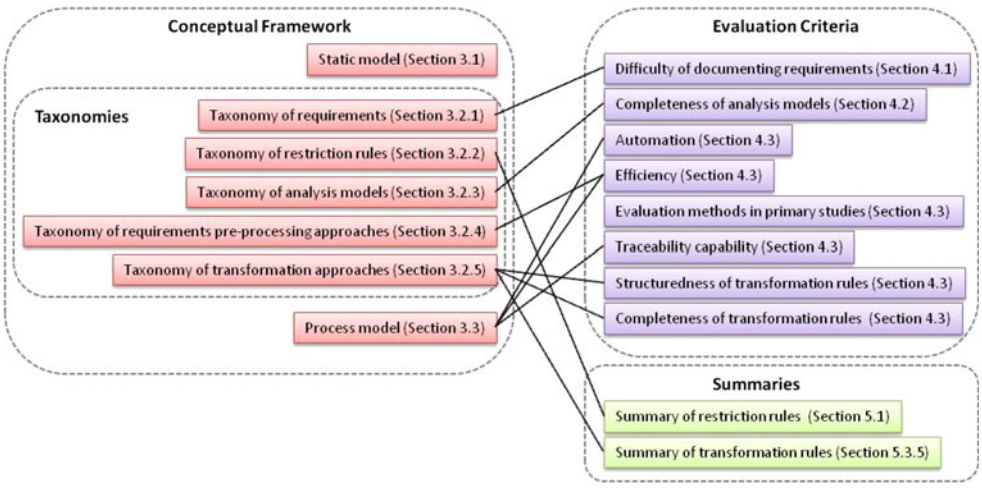

Fig. 8 Mapping between the conceptual framework and evaluation criteria and summaries

Table 2 Evaluation summary of the approaches proposed in the primary studies

Requirements configuration[a]		Analysis model[b]	Pre-processing (Step 1)[c]	Steps (Fig. 7)	Transformation[d]	Automation	Efficiency	Primary studies
#	(DSI, Representation, Restr. NL?)							
1	(None, None, No)	Object diagrams	LA, SynP, SemP, PA	(1, 2, 4)	(R, R)	Automated	Low	[36]
		Classes, attributes, and associations	LA, SynP, SemP	(1, 2, 4)	(O, R)	Automated	Low	[21]
		Domain models, hybrid activity diagrams	LA, SynP	(1, 2, 3, 4)	(R, R, R)	Automatable	Low	[24]
		Class, activity, state machine diagrams	LA, SynP, SemP	(1, 2, 4)	(R, R)	Semi-automated	Low	[18, 35, 44]
		Architecture concepts (e.g., components and connectors).	Catg	(1, 2, 4, 5)	(R, R)	Manual	N/A	[20]
		Class diagrams	LA, SynP	(1, 6)	(R)	Semi-automated	N/A	[40]
		Class diagram, coarse-grained behavioral concept	Catg	(1, 2, 4)	(R, P)	Manual	N/A	[10]
		Data types, variables, operations, control constructs (e.g., *if–then-else* and *for* loop)[e]	None	(6)	None	Manual	N/A	[4]
2	((Glossary, Definition), None, No)	ERM, DFG, UML models (not described)	LA, SynP, SemP	(1, 2, 3, 4)	(I, R, R)	Automated	Low	[6]
3	(None, OBFS, Yes)	Class diagrams	LA, SynP, Catg	(1, 6)	(R)	Automatable	Low	[54]
4	(None, Use cases, No)	Sequence diagrams	Unknown	(?, ?, 4)	(?, P)	Automated	Unknown	[14]
		Message sequence charts	Catg, LA, SynP	(1, 6)	(R)	Automated	Low	[17]
		Class and sequence diagrams	None	(5, 6)	N/A	Manual	N/A	[25]
5	(None, Use cases, Yes)	Extended sequence and extended activity diagrams[f]	None	(6)	(P)	N/A	N/A	[47]
6	(Glossary, Use cases, Yes)	Class diagrams	LA	(1, 2, 3, 4)	(R, R, R)	Automated	Low	[34, 52]
7	(Domain model, Use cases, Yes)	State machines	None	(6)	(R)	Automated	High	[45, 49]

[a] The requirements configurations requiring more significant effort to document requirements are highlighted with a darker color

[b] The approaches that are capable of generating analysis models, i.e., including both structural and behavioral aspects of a system, are highlighted

[c] *LA* lexical analysis, *SynP* syntactic parsing, *SemP* semantic parsing, *Catg* categorization, *PA* pragmatic analysis, *None* means that the corresponding approach does not need requirements pre-processing

[d] *R* rule based transformation, *O* ontology based transformation, *P*, pattern based transformation, *I* identity transformation

[e] The heuristics of [4] have been adapted in Software Engineering textbooks (e.g., [9, 32]) to generate UML class diagrams as analysis models

[f] The approach extends UML sequence and activity diagrams to represent requirements including some concepts of use case models (e.g., precondition)

4.2 Evaluation criterion for analysis models

We need to evaluate the generated analysis models with respect to their completeness. From a user's perspective, a generated analysis model is expected to be as complete as possible so that it can be a useful starting point of an iterative analysis refinement process. Various types of analysis models can be generated by the approaches proposed in the primary studies, such as UML models [54] and MSCs [17]. Though it is difficult to compare different types of analysis models, their common modeling capabilities can be extracted and used as the basis for comparison. For example, MSCs can be considered similar to UML sequence diagrams, because both model the dynamic behavior of the system in terms of object interactions through messages. If a generated analysis model both describes the system structure (e.g., class diagram) and behavior (e.g., sequence diagram, state machines, or activity diagrams), then we label the generated analysis model as complete. If a generated analysis model describes only one of these two aspects of a system (i.e., either the structure or behavior), then we label it as incomplete.

4.3 Evaluation criteria for transformation

We evaluate the transformation approaches proposed in the reviewed approaches with respect to their automation and efficiency. The automation criterion evaluates whether a transformation is automated, automatable, semi-automated, or manual. A transformation approach is automated if it has been fully implemented. If a transformation algorithm is proposed in a paper, then we assess whether we deem the description is sufficient to implement it, and if this is the case, the transformation approach is deemed automatable. In some cases, a transformation is semi-automated because user interventions are required. Last, some approaches are entirely manual.

The efficiency of an approach is evaluated by analyzing how many transformation steps are necessary, and how many requirements pre-processing techniques (Sect. 3.2.1.2) are applied. If it takes several transformation steps for an approach to transform requirements into an analysis model, we label this approach's efficiency as low, as opposed to an approach requiring only one single step. If an approach needs three or more requirements pre-processing techniques, we also label it as having a low efficiency.

Extensive case studies are a necessity since validating transformation approaches cannot be performed in an analytical way. Selecting case studies to run and how results are analyzed are two important aspects of the evaluation of an approach. We evaluate each approach and examine: (1) the number and size of the case studies performed, (2) the results of the case studies reported, and (3) whether other evaluation approaches (besides case studies) are described.

Traceability links between requirements and analysis model elements are expected to be established when a transformation is performed. Because only a few of the reviewed approaches report on traceability, we only examine whether traceability is reported in each approach and do not analyze the details of the traceability link generation strategies.

Transformation rules specify which requirements constructs map to which analysis model elements. They are expected to be complete and well-structured. If, according to our understanding to the transformation rules as they are described in primary study papers, the transformation rules proposed in an approach can transform most or all requirements constructs into analysis models elements, then we say this set of transformation rules is complete; otherwise, incomplete. We expect each approach to evaluate the completeness of its transformation rules by for example, performing case studies. However, some of the approaches do not evaluate the completeness of transformation rules and some of them do not even describe transformation rules. This simply makes the evaluation impossible. If transformation rules are presented in the primary studies and organized according to the structure of the source language, the target language, some other relevant organization (e.g., rule composition), or different transformation phases [12], and each transformation rule is well specified (e.g., using a carefully defined language like OCL), then we label the transformation rules as well structured.

4.4 Discussion

One may argue that it is possible to perform a finer-grained, more objective analysis such as evaluating how restrictive each restriction rule is if restricted NL is applied, how complete is each aspect (or diagram) of generated analysis models (e.g., amount of information generated in the class diagram), how efficient is each pre-processing technique and each transformation step. However, it is difficult (if not impossible) to perform such an analysis because: (1) No sufficient information is provided in the primary studies (e.g., in many cases no case study is presented and the completeness of their transformation rules is not discussed, when they are described in detail); (2) Empirical studies are needed to perform a finer-grained analysis to evaluate the restrictiveness of each restriction rule and the overall completeness of each diagram, which is out of the scope of this paper; (3) Some approaches are manual; therefore, the completeness and correctness of generated analysis

models and their overall efficiency strongly depend on the capability of users; (4) Different types of diagrams are generated and therefore it is difficult to have common evaluation criteria for evaluating the completeness of generated analysis models; (5) It is common for primary studies to use different case study systems and therefore it is hard to have objective evaluation criteria for the completeness of the generated analysis models.

Our evaluation criteria, though coarse-grained, are still sufficient to differentiate each approach and are straightforward to apply, thanks to the well-specified conceptual framework and the clear mapping between it and the evaluation criteria. Furthermore, as illustrated by the results of the comparison, no such fine-grained analysis was required to compare approaches: our criteria are precise enough to allow us to differentiate different approaches.

5 Synthesis and evaluation

In Sect. 3.2, we defined taxonomies to classify and characterize techniques and concepts used in the primary studies. The selection of one or more than one element from each of these taxonomies is denoted as a *configuration* characterizing a given approach. Such configurations are a way for us to abstract away from details and allows the analysis of emerging, general patterns. The taxonomy of requirements contains three sub-taxonomies: DSI, Representation, and Natural Language. A combination of one or more element from these three sub-taxonomies forms a *requirements configuration*. For example, if an approach does not require DSI, is based on use cases which are described in restricted NL, then the requirements configuration of the approach is presented as a tuple (*None, Use Case, Yes*). As shown in Table 2, the approach proposed in [47] conforms to this requirements configuration (configuration 5 in Table 2, Column 2). Steps taken by each approach are different. For example, some approaches (e.g., [40]) that do not contain intermediate models but require requirements pre-processing contain only Step 1 and Step 6 (Fig. 7), presented as a tuple (*1, 6*) in Table 2, Column 5. If an approach contains two transformations (one intermediate model), the transformation from pre-processed requirements to the intermediate model is rule-based, and the transformation from the intermediate model to analysis models is also rule-based, we use a tuple (*R, R*) to represent the configuration of the transformations, as shown in Table 2, Column 6. Over all, an *approach configuration* is characterized by a requirements configuration, analysis models, requirements pre-processing techniques, steps taken by the approach, types of transformations, and automation and efficiency of the approach. Configurations of each approach are given in Table 2, grouped by requirements configurations.

In the rest of the section, using the notion of approach configuration to structure the discussion and abstract away from minor differences, we first analyze and evaluate the reviewed approaches in terms of requirements configurations, analysis models, and automation and efficiency of transformations. Next, we evaluate the reviewed approaches from other, complementary aspects such as whether an approach is evaluated and whether the evaluation is properly described in its primary study. Last, we summarize the evaluation results. The detailed analysis of each primary study is provided in [57] for reference.

5.1 Requirements configurations

A total of seven different configurations match the reviewed approaches (Table 2, Column 1). Configuration 1 requires no DSI, no specific representation, and no restricted NL. This configuration is the most frequently used one; eight out of 16 approaches comply with this configuration. Configurations 2, 6 and 7 require a DSI (Sect. 3.2.1) as part of their requirements input, to assist the computational NL processing. For example, a glossary is mainly used to identify entities, objects, or classes. A domain model serves as the structural basis of target models such as sequence diagrams. Most of the domain specific information is manually constructed. The rest of the configurations do not require any DSI.

Configuration 3 needs requirements to be documented using the OBFS template, a customized document template that the transformation technique of the approach [54] relies on. This configuration does not need any DSI or restricted NL. Configurations 4–7 (six approaches) take use cases as requirements representation. This is reasonable since use cases are a commonly used notation for capturing requirements in practice. Besides, a use case template helps organize textual requirements so that the requirements preprocessing and the following transformation(s) can be facilitated.

Configurations 3, 5–7 require that requirements be documented using restricted NL. There are three main reasons why restricted NL is used in requirements documentation. A restricted NL aims to reduce ambiguity, redundancy, and complexity in documents. It also makes computational NL processing more reliable, efficient, and accurate. Last, it facilitates translation into other languages. However, the extent of restrictions varies across approaches and a balance should be struck between the applicability of restriction rules and facilitating analysis. We summarize the restriction rules applied in the primary studies (i.e., [47, 49, 52, 54]) in Table 3. These rules are

Table 3 Restriction rules on requirements documentation

Restriction	Restriction rules	Applying situation	Purpose	Rel. works
Sentence restriction	Apply simple sentence[a]	Any statement	Facilitate automatic NL parsing; reduce ambiguity; simplify the complexity of sentences	[47, 54]
	Use active voice rather than passive voice (actor is omitted)	Any statement	Facilitate automatic NL parsing; easier to identify messages or behavior	[52]
	Use the same verb for the same action in different sentences	Use case → flow of events	Improve the quality of NL parsing; reduce ambiguity	[52]
	Do not use pronouns	Any statement	Facilitate automatic NL parsing	[52]
Sentence structure restriction	And, or	Use case → condition	Specify composite conditions	[49]
	GO TO Step [number]	Use case → Branching statements	Specify branching	[49]
	CON [statement]	Concurrency statements	Specify concurrency statements	[52]
	If–then	Conditional statements	Specify conditional statements	[52, 54]
	While-endWhile, Repeat[number]until[states], Do-until	Iteration statements	Facilitate the transformation to sequence diagram	[52]
Wording restriction	AFTER [duration], BEFORE [duration]	After delay and before delay statements	Facilitate the transformation to state machines (timeout transitions)	[49]
	AND ON [entity]	Use case → Postcondition	Facilitate the transformation to state machines	[49]
	Is a kind of, is specialization of, is generalization of	Inheritance sentences	Identify generalization between subject and object	[54]
	Drive, work for, maintain, manage, own, execute, serve, use	Action sentences	Identify objects and associations	[54]
	Talk to, communicate with, refer to	Communication sentences	Identify objects and associations	[54]
	Next to, goto	Location sentences	Identify objects and associations	[54]
	Has (a capability of), has (a capacity for), can, able to has not (a capability of), has not (a capacity for), cannot, not able to	Behavioral sentences	Identify behaviors	[54]

[a] A simple sentence is composed of one subject and one predicate

classified into three categories: sentence restrictions, sentence structure restrictions, and wording restrictions (Sect. 3.2.1.1). The table also indicates where rules are applicable and their purpose. Examples are given for some rules. The restricted NL used in [47] is not well described in the paper. That is why only one restriction rule is presented in the table.

Based on the data we extracted from each primary study and summarized in the first column of Table 2, we discuss next how difficult it is to document requirements in the format required by a specific approach according to our evaluation criteria (Sect. 4.1). The configurations requiring more significant effort to document requirements are highlighted with a darker color, following the rationale described next. The evaluation results show that it is most difficult to document requirements in the format required by configuration 7 (approach proposed in [45, 49]) because a great deal of user effort is needed to obtain a domain model containing classes, associations, and operations, which are indispensable for generating state machines, and additionally use cases are required to be written in restricted NL. Configurations 2, 3, and 6 require the second largest effort to document requirements. Though configuration 2 [6] does not rely on restricted NL and does not require any specific requirement representation, the difficulty of documenting requirements is still high as users are

required to manually specify glossary and a significant number of definitions in a specific form. Configuration 3 [54] implies requirements to be manually documented in a non-standard modeling language (OBFS) and the use of restricted NL. Configuration 6 [34, 52] needs a glossary, and use cases are required to be written in restricted NL. Configuration 5 requires even less user effort since no DSI is necessary. Configuration 4 requires less effort than Configuration 5 to document requirements as not only no DSI is required but additionally use cases do not need to be documented using restricted NL. Configuration 1 requires the least effort to document requirements.

5.2 Analysis models

We can see from Table 2, Column 2 that twelve out of 16 approaches can derive structural model elements (e.g., objects, classes, associations, components) from requirements. Most of the approaches are able to generate objects, classes, and associations, but not all of them can generate attributes, operations, and generalizations. Nine approaches can generate behavioral features of a system (e.g., sequence diagrams, state machines, and/or activity diagrams).

Three approaches ([24], [18, 35, 44] and [10]) (highlighted) conforming to configuration 1 are capable of generating analysis models including both structural and behavioral aspects of a system, which are characterized as complete according to our evaluation criteria (Sect. 4.2); two approaches ([6] and [25]) (highlighted) conforming to configuration 2 and 4, respectively, can also generate complete analysis models. The generated domain models of the approach proposed in [24], conforming to configuration 1, contain only objects and links, rather than commonly used class diagram representations; the generated hybrid activity diagrams (i.e., UML activity diagrams also including the concepts of actors, business rules, and messages) are at a very high level of abstraction, and are independently generated from the generated domain models (i.e., there might be inconsistencies between the two diagrams). The NIBA project [18, 35, 44], also conforming to configuration 1, can derive class, activity, and state machine diagrams from requirements. User intervention is required in many places, especially during the transformation from requirements to intermediate models. There is not enough information provided in the papers to show that the generated class, activity, and state machine diagrams are correct, consistent, or precise enough. The approaches proposed in [10] and [25], conforming to configurations 1 and 4, are all manual; therefore the completeness and correctness of generated analysis models mainly depend on the capability of users, rather than the approaches themselves. The approach proposed in [6] requires a great deal of user effort on documenting requirements, two intermediate models (three transformations), and a sequence of requirements pre-processing techniques.

Not surprisingly, UML (e.g., class, activity, sequence, and state machine diagrams) is the most frequently used language in the reviewed approaches to represent generated analysis models.

5.3 Transformation–automation

Only five approaches describe the algorithms they used to various extents of details. Most of these algorithms are not described at a level of detail that is amenable to an implementation. According to the evaluation criterion discussed in Sect. 4.3, we summarize the evaluation results of transformations: automated, automatable, semi-automated, or manual.

As shown in Table 2, Column 7, seven out of 16 approaches are automated; two are not automated but are automatable; two approaches require user intervention to semi-automatically perform the transformation; four approaches require manual transformations. Complex pre-processing techniques are required for all the automated approaches, except the approach proposed in [34, 52] (Configuration 6), which only requires lexical analysis, and the approach proposed in [45, 49] (Configuration 7), which does not have any requirements pre-processing techniques since the transformation from use cases plus a domain model to state machines relies on the template structure of the use cases and the domain model. However, two intermediate models (three transformations) are required in this approach (Column 5). For the approach proposed in [14] (first approach in Configuration 4), the transformation from use cases to intermediate models (Step 2) is not described in the paper and therefore the automation of this step is unknown as indicated in the table. The approaches proposed in [45, 49] and [17] have been implemented and therefore they are automated approaches. The one proposed in [54] is not automated but is automatable, and the one proposed in [40] is semi-automated since a significant user intervention is required. The transformation is not explicitly discussed in [47], because the approach does not attempt to provide a solution for the transformation of requirements into analysis models though the proposed approach can be adapted to that purpose, which is also the reason why we included this paper for review. Last, three manual approaches are proposed in [25], [10] and [4], respectively. Though the approach proposed in [45, 49] is automated, a great deal of user effort is needed to obtain a domain model and specifying use cases and applying restrictions. Additionally, the consistency between the domain model and the use cases must be manually maintained. Manual requirements pre-processing (e.g., users are required to manually classify the sentences) is required for the automated approach proposed in [17].

5.4 Transformation–efficiency

As we have discussed in Sect. 4.3, the efficiency of an approach is evaluated by analyzing how many transformation steps are taken in the approach, and how many requirements pre-processing techniques are applied.

As shown in Table 2, Column 4, most approaches apply at least one of the requirements pre-processing techniques (Sect. 3.2.1.2). We do not know what requirements pre-processing techniques are applied in the approach proposed in [14], since it is not described in the paper. The approach proposed in [45, 49] does not have any requirements pre-processing technique since the transformation from use cases plus a domain model to state machines relies on the template structure of the use cases and the domain model. The approach proposed in [47] does not require any requirements pre-processing technique because the approach describes three equivalent requirements representations, and each of them can be transformed into the other. The approach proposed in [25] does not need any requirements pre-processing technique since it proposes a set of techniques for users to manually specify requirements and also a process to guide the users to derive the conceptual models from the requirements. It does not aim to automatically transform requirements into an analysis model. Similarly, Abbott's heuristics [4] do not need any requirements pre-processing technique.

As shown in Table 2, Columns 5 and 6, rule-based transformations are most frequently used to create the first intermediate model (Step 2 of the process): first letter in the transformation tuple (Column 6). An ontology-based transformation is used in [21] since the intermediate model is all ontology-based. A "?" for approach [14] indicates that the transformation is unknown since it is not discussed in the paper. Only one approach [47] applies pattern-based transformations (denoted as "P") and only one approach [6] applies identity transformation (denoted as "I"). Most of the approaches containing Step 4 use rule-based transformations (except [10] and [14]). Eight approaches use intermediate models (containing Step 2), when direct transformation from requirements to an analysis model cannot be achieved. Two intermediate models (three transformations, and therefore Step 3 is required) are contained in [6], [24], and [34, 52] instead of only one intermediate model (two transformations) in the other six approaches that use intermediate models. Most of the approaches use rule-based transformations to transform pre-processed requirements directly into an analysis model (Step 6).

According to our evaluation criterion on efficiency of approaches, the approach proposed in [45, 49] shows highest efficiency because it does not need any requirements pre-processing technique and requirements are directly transformed into analysis models. Note that it does not make sense to evaluate the efficiency of manual approaches so their efficiency is marked as "N/A".

5.5 Transformation–others

5.5.1 Evaluation

Only four out of 16 approaches have their transformation approaches evaluated. Case studies have been performed to evaluate the approaches proposed in [21] and [14] by manually comparing the tools results with the manually constructed analysis models. A performance evaluation method is also proposed in [21] and five case studies were performed to evaluate the performance of the tool. The evaluation results show that the approach can perform better than other language-processing technologies, such as information retrieval systems. Three industrial pilot studies were performed to test the acceptability of the tool implementing the approach proposed in [6]. The evaluation of the approach proposed in [18, 35, 44] is not discussed in details in the papers, except for the statement that "the approach has been applied for practical requirements analyses and the results showed to be encouraging." The other approaches were not evaluated, though some of them present a running example to illustrate their approach rather than to evaluate it.

5.5.2 Traceability support

Among the papers we have reviewed, only two transformation approaches [20] and [25] report on traceability. In [20], it is claimed that traceability is supported, though this is not discussed in the paper. A traceability model, represented as a function decomposition table (rows are use cases and columns are the identified classes), is proposed in [25] to link the identified classes to the use cases. Deriving traceability links (Step 7) from already established links is not an issue for transformation approaches that do not involve intermediate models; however, for those which require one or more intermediate models, it is an indispensable step since from the users' perspective it is very important to access derived traceability links between requirements and analysis models without having to deal with the intermediate model(s). This step is not covered in any of the approaches we reviewed.

5.5.3 Completeness and structuredness of transformation rules

Nine ([36], [21], [34, 52], [24], [18, 35, 44], [10], [17], [54], and [4]) out of 16 approaches describe their transformation rules in their primary studies but none of them

evaluate the completeness of the transformation rules. Five ([6], [14], [20], [45, 49] and [40]) out of 16 approaches do not describe their transformation rules at all. Note however that the completeness of the transformation patterns of [14] was evaluated by performing some case studies (not described in the paper though). The evaluation was manually performed by comparing the tool generated interaction models with the ones manually constructed by the experts. The evaluation results show that 65% of the sequence diagram fragments generated by the tool are identical (i.e., modeling the same interactions with the same instances and the same messages) to the manually obtained sequence diagram fragments, 28% of the automatically generated fragments are equivalent (i.e., modeling the same interactions with different instances and messages) to the manually obtained one, and 7% of these fragments are different (modeling different interactions). The approaches proposed in [25] and [47] do not purport to provide solutions for transforming requirements into analysis models; though both of them can be adapted to that purpose, which is also the reason why we included them. Therefore, no transformation approach is discussed in these two papers.

Seven approaches directly transform requirements into analysis models (Step 6): [40], [54], [17], [25], [47], [45, 49], and [4]. The others use intermediate models to bridge the gap between requirements and the analysis model. Transformation rules of these indirect transformation approaches contain two rule sets: transformation rules from requirements to intermediate models and transformation rules from intermediate models to the analysis model. The intermediate models act as the target models of the first rule set and also the source models of the second rule set. Because of the differences among these intermediate models, it is hard to synthesize these rules. Therefore, we only summarize and synthesize the transformation rules of the rule-based transformation approaches that directly transform requirements into an analysis model, except for the approach proposed in [47] in which the transformation is not explicitly discussed, because the approach does not aim to provide a solution for the transformation from requirements into an analysis model though the proposed approach can be adopted to achieve that. The papers [45, 49] and [40] do not describe the transformation rules they used. The transformation rules from [17], [54], [25], and [4] are presented in Table 4 and Table 5. We also summarize (in the same tables) the heuristics rules proposed in [9, 31, 33] which extend or do not refer to Abbott's heuristics rules [4]. Their completeness, effectiveness, and correctness are not evaluated through empirical studies. Though the approach proposed in [21] does not directly transform requirements into an analysis model (intermediate model is required), the paper describes the mapping relations between the two types of transformation rule sets and therefore the mapping relations are derived as transformation rules and also included in the Table 4 and Table 5. In each one of these four approaches, transformation rules are independent from each other: Each rule simply describes the mapping relationship between a requirements concept (Column 2) and an object-oriented concept (Column 3). Requirements constructs include natural language concepts (e.g., noun, subject, etc.). In Column 4, constraints are provided when necessary. Column 6 provides some examples for the transformation rules that are not easy to understand.

5.6 Summary of evaluation results

An ideal approach for transforming requirements into analysis models would have the following characteristics: (1) requirements should be easy to document using the format required by the approach, (2) generated analysis models should be complete (i.e., contain structural and behavioral aspects of a system), (3) the approach should contain the least number of transformation steps as possible (high efficiency), (4) the approach should be automated, and (5) the approach should support traceability management (Step 5). However, none of the reviewed approaches conforms to the ideal configuration, as described next.

1. Requirements configuration
   a. Requirements configuration 1 (Table 2)
      The approaches conforming to requirements configuration 1 require the least user effort to document requirements. However, only two of these approaches are automated and one is automatable. The other five approaches are either semi-automated or completely manual. Besides, complicated requirements pre-processing techniques and intermediate models are required for the two automated approaches and therefore their efficiency is low. It is also worth noticing that these two automated approaches are not capable of generating complete analysis models, i.e., including both static and dynamic aspects of a system.
   b. Requirements configuration 4
      Requirements configuration 4 ranks second in terms of user effort to document requirements. Two of the three approaches conforming to this configuration are automated, which however can only automatically generate the behavioral aspect of a system, instead of a complete analysis model.
   c. Requirements configuration 5
      Compared with requirements configuration 4, requirements configuration 5 requires use cases to be documented in restricted NL; therefore, it

Table 4 Summary of transformation rules (part 1)

Transformation rule			Rel. work	Example
Requirements concepts	OO concepts	Constraint		
(recurring) noun or noun phrase	Object, class		[4, 9, 21, 33]	
Subject of a sentence	Object, class	The subject is noun	[17, 54]	
Object of a sentence	Object, class	The object is noun	[17, 54]	
Actor of use cases	Object		[9, 25]	
Use case	Object	≪control≫ object	[9]	
Genitive case (e.g., using *of*, *'s*)	Attribute	The first noun is the attribute of the second noun	[9, 21]	The `name` of a student
The object (noun) of a simple sentence	Attribute	The predicate of the sentence contains *has* | *consist of* | *contain of* | *denote* | *identify*	[21, 54]	Person has `name`
Attributive adjective	Attribute value of the noun that the attributive adjective modifies		[4, 21]	A large library has many sections. '*large*' is the value of the attribute `size` of the class `Library`.
Doing verb	Operation		[4]	'*submits*' are doing verbs
Having verb	Aggregation		[4]	'*has*' and '*consists of*' are having verbs
Verb/verb phrase	Association	Verbs/nouns connecting two objects	[9, 31]	Two trains following each other. *'following'* is the verb connecting two objects; therefore a reflexive association is identified for class `Train`.
Property sentences	Aggregation association	*Is made up of* or *is part of* or *contains* is used in the sentence.	[21]	The `university` contains 10 `departments`.
Universal quantifier (first entity) + unique existential quantifier (second entity)	Many-to-one association	From the first entity to the second	[21]	A complex `aircraft` uses the `radar`.
Singular (first entity) + quantified by the definite article (second entity)	One-to-many association	From the first entity to the second	[21]	The `student` passed all `exams`.
Singular (first entity) + singular (second entity)	One-to-one association	From the first entity to the second	[21]	The `student` passed the `exam`.
Specific number	Multiplicity	Specific number	[21]	The `student` passed 3 `exams`.

requires more user effort to document requirements. The approach conforming to this configuration still cannot generate complete analysis models.

d. Requirements configurations 2, 3 and 6
Two approaches conforming to requirements configurations 2 and 6, respectively, are automated and the approach conforming to configuration 3 is automatable. Requirements needed by these three configurations rank second in terms of documentation difficulty. Only one of them (i.e., [6]) is capable of generating a complete analysis model. Additionally, the efficiency of the approach is low since two intermediate models (three transformations) and a sequence of requirements pre-processing techniques are needed.

e. Requirements configuration 7
Requirements configuration 7 is the one that requires the most user effort to document requirements. The approach is automated and does not

Table 5 Summary of transformation rules (part 2)

Transformation rule			Rel. work	Example
Requirements concepts	OO concepts	Constraint		
Being verb	Inheritance/generalization		[4]	'*is a kind of*' is a being verb
Modal verb	Constraints		[4]	'must be' is a modal verb
Verb/verb phrase	Behavior	Verb, predicate contains *has a capability to* | *can* | *able to*	[4, 54]	The student has a capability to learn. `to learn` is the behavior of `the student`
Direct object	Message	Sentence structure like *subject-direct object-indirect object*	[17]	The clerk sends the `status of the load_bay` to the system
Transitive verb	Message	Sentence structure like *subject-transitive verb-object*	[17]	The attendant `enables` the pump
Basic flow, alternative flow of use cases	Sequence diagram		[25]	

need requirements pre-processing. The efficiency of the approach conforming to this configuration is high. However, the approach still cannot generate complete analysis models (i.e., static and dynamic aspects).

f. As expected, approaches requiring more user effort to document requirements achieve better automation and higher efficiency.

g. Use cases are the most frequently applied requirements representation.

2. Analysis model representation
UML diagrams are the most frequently used representations of analysis models, which confirms that in practice UML is used in many IT software development organizations [41].

3. Efficiency

   a. Requirements pre-processing
   Most of our reviewed approaches apply at least one of the requirements pre-processing techniques, among which lexical analysis (Sect. 3.2.1.2) is the most commonly used technique. This is understandable because requirements are usually written in textual form that must be tokenized and POS of sentences should be identified in order to facilitate transformations. Syntactic parsing (Sect. 3.2.1.2) is also commonly applied in the approaches that require determining grammatical structures such as subjects and predicates of sentences. When not applying any pre-processing technique (except categorization), one needs to manually transform requirements into intermediate models or analysis models. Categorization (Sect. 3.2.1.2) is another technique frequently used in the primary studies. All these papers require categorization to be performed manually. Complex pre-processing techniques are usually required for automated approaches.

   b. Transformation steps
   Most of the reviewed approaches have one intermediate model. Few of them need two intermediate models. For those using intermediate models (containing Step 2), rule-based transformations are most frequently used.

   c. Efficiency
   According to our evaluation criterion on efficiency of approaches, only one of the reviewed approaches [45, 49] has clearly superior efficiency because it does not need any requirements pre-processing technique and requirements are directly transformed into analysis models.

4. Automation
More than half of the reviewed approaches are automated or automatable. A high level of automation is an absolute requirement for any approach to scale up in industrial practice.

5. Approach configuration
No approach, with acceptable user documentation effort and efficiency (e.g., one or two transformation steps), is currently able to automatically or semi-automatically generate a complete (i.e., containing both static and dynamic aspects), consistent analysis model.

6 Open issues and suggestions

As we have discussed in Sect. 5, a desirable approach, involving acceptable user effort in documenting requirements, should be able to (semi-) automatically and efficiently generate a complete (i.e., including both static and

dynamic aspects of a system) and consistent analysis model. Since none of the existing approaches achieves this, based on the systematic review results, the goal of this section is threefold. We want to identify recurring issues in the research and reporting of the primary studies we reviewed, highlight open issues in existing solutions, and identify useful avenues of research.

6.1 Approach configuration

In this section, we first discuss the open issues identified for each aspect in an approach configuration. Then we recommend an approach configuration which, with due research, should be able to provide a solution to automatically and efficiently generate complete analysis models, based on acceptable user effort in documenting requirements.

6.1.1 Requirements configuration

A desirable requirements configuration should be able to effectively facilitate transformation from requirements to analysis models, while minimizing user effort in documenting requirements. However, tradeoffs exist between the difficulty of following a requirements configuration and the extent to which it facilitates transformation, especially automated transformation:

(1) Some approaches require additional DSI (Sect. 3.2.1) as requirement supplements; however, these approaches rely on users to manually provide DSI so that a great deal of user effort is required. We believe that demanding a textual glossary as a requirements supplement could be practical and requiring a domain model or definition could lower the representation gap between requirements and analysis models. However, it would be desirable to generate such a domain model automatically from requirements, at least an initial version to be refined, rather than asking users to provide it. Furthermore, if the modeling of DSI is required, this should be well supported by tools.

(2) Other approaches do not use any representation to structure their requirements (i.e., pure textual specifications); however, if requirements are structured (e.g., using use case templates), one can expect that transformations be greatly facilitated. Almost half of the approaches require anyway that their requirements be documented using some form of use case template. Besides, use case modeling is commonly applied in practice. Therefore, we suggest having use cases, using appropriate templates, as the means of documenting requirements to facilitate automated transformations. Whether a use case template is easy to apply and whether it is able to effectively facilitate automated transformations should be experimentally investigated [56].

(3) Restricted NL is sometimes used for documenting requirements; however, the rationale for restriction rules is often not clearly justified. Our summary table regarding restriction rules (Table 3) provides the rationale of each rule (Column 4 of the Table), but we had to devise them by carefully examining each primary study since this information was in most cases not provided. It is important to know why a particular restriction rule is applied because further research may relax it by, for example, using new or improved NL analysis techniques. It is also paramount to know whether a set of restriction rules is easy to apply and whether its application can lead to a higher quality of automatically derived analysis models [56]. Again, experimental evaluations are required to further investigate this issue.

In summary, we believe that (i) it is desirable not to require additional DSI, though it may be practical to demand a textual glossary, (ii) use cases should be supported as they are most frequently used for requirements representation, and (iii) restricted NL might be used for documenting requirements so that automated transformation can be facilitated. Using our tuple representation—(DSI information, requirement representation, NL requirement)—we therefore, recommend that the following set of requirements configurations be considered in future work: (*None, Use cases, No*), (*None, Use cases, Yes*), (*Glossary, Use cases, Yes*), and (*Glossary, Use cases, No*). We also recommend that experimental evaluations be performed to evaluate a requirements configuration method in terms of its applicability and effectiveness at automatically deriving analysis models.

It is worth noticing that requirements are not stand-alone artifacts; goals, assumptions, standards, and risks are all part of a complete requirements document. However, in the context of MDA, in order to generate an object-oriented analysis model, object-oriented analysis and design methodologies mostly use functional requirements as input for this specific transformation. Higher-level requirements artifacts such as goals, assumptions, standards and risks usually form a basis and justification for deriving detailed functional requirements (represented as use cases), which can then be further used to derive analysis models. Most of the primary studies identified by our systematic review take functional requirements (represented as use cases) as input to generate analysis models (Table 2) but the rest only use unstructured text as input (e.g., [36] and [24]), therefore not specifying the type of requirements they use in input.

6.1.2 Analysis model

As we have observed in Sect. 5.2, UML diagrams are most frequently used in the reviewed approaches to represent analysis models. This conforms to the MDA [29] transformation concept, which requires the source (e.g., PIM) and target (e.g., PSM) of a transformation to be represented as UML models. UML is a standardized language, is widely supported by a growing body of tools (e.g., [22]), open source plugins (e.g., [16]), and has been specialized for many domains.

If use case models, including use case diagrams and use case specifications, are used to structure and document requirements and UML models are used as the representation of the analysis model, a relationship can be clearly established between the use case models and parts of the analysis models. In particular, since use case descriptions describe interactions of the system and actors along the time line, they can be transformed into messages in sequence diagrams, an important component of behavioral modeling in analysis models. With an appropriate use case template, it is expected that conditions and branches in use case specifications can be automatically captured and transformed into *CombinedFragment*s [38] in sequence diagrams. In addition, extend and include relationships in use case specifications can be transformed into *InteractionUse* [38] of sequence diagrams.

UML models can model not only the structural aspect of a system (e.g., class diagrams), but also the behavioral aspects (e.g., sequence and activity diagrams). Though this is to some extent dependent on the modeling method used, consistency between the structural and behavioral aspects can be easily achieved in the context of UML since when transformations are performed, one single UML model is created, queried, and maintained during the transformations; different diagrams are just different, overlapping views of the same underlying model.

Therefore, for the above practical and technical reasons, we suggest using UML models as the representation of analysis models.

A methodological open issue we identified in this review is that many of the approaches cannot generate a complete analysis model (i.e., both structural and behavioral aspects). Additionally, the correctness of their generated analysis models is not evaluated. The quality of an automatically generated analysis model should be evaluated by, for example, comparing it with existing expert solutions to see how close the automated analysis model is to these expert solutions.

6.1.3 Automation

The level of automation is one of the important characteristics of transformations. Automated transformations are always desired; however when a certain amount of manual intervention is indispensable for documenting requirements, performing transformations, or establishing traceability links, it should be explicitly described and its expected effort should be evaluated. For automated approaches, transformation algorithms should be clearly specified, and this is a requirement which is not always met in the approaches of this review.

6.1.4 Efficiency

Our evaluation results show that most of our reviewed approaches need complicated requirements pre-processing, contain two or more transformation steps, and/or user intervention is required in many places. In terms of requirements pre-processing techniques, some approaches require significant user effort to manually pre-process requirements, for example, manual categorization of requirements (Sect. 3.2.1.2). We suggest that only automatable NL processing techniques should be used. Since it is paramount to automate transformations, user's involvement should be minimized. Additionally, the more intermediate models, the more difficult the validation and verification of the approach; the more intermediate models, the higher the chances of loosing information during the transformation from requirements to analysis models (because of multiple transformations). However, the complexity of transformations and amount of information to manipulate suggest that not relying on an intermediate model might be difficult to achieve. Indeed, most of our reviewed approaches have one intermediate model (two transformations). Last, we will argue in Sect. 6.3 that one intermediate is necessary. Therefore, we suggest that a maximum of one intermediate model be required in an approach.

According to above discussion, we recommend the following set of approach configurations:

> Automatically transform use case models with or without restricted NL and/or glossaries to complete (i.e., including both static and dynamic aspects), correct and consistent UML models using one intermediate model and fully automatable requirements pre-processing techniques (e.g., lexical analysis and syntactic parsing).

6.2 Intermediate model

Some approaches use intermediate models as bridges for transformation between requirements and analysis models. The main reason is that requirements are usually text-based, and automated transformations (to fully integrate requirements into model-driven approaches) cannot be

easily supported with unrestricted, unstructured requirements representations such as pure text. The reason for using one specific type of intermediate models should be explicitly justified in the research literature, and the following considerations should be taken into account when intermediate models are selected:

- The representation of the source and target models since they drive the selection of intermediate models (if any) as well as transformation rules.
- Whether the intermediate model(s) can be easily integrated into existing tool support.
- If user interventions are required during transformations, it is important that the intermediate model be easy to understand by users.
- Whether the intermediate model is general enough to be used for multiple purposes, such as generating not only class diagrams, but also sequence diagrams, activity diagrams, and state machines. The intermediate model KCMP [18] is one such example.
- Whether it can be used independently of different NL processing techniques.
- Whether it is suitable to support traceability analysis.

The above items are usually not carefully discussed in most primary studies. As a result, the proposed technologies are often difficult to assess.

6.3 Transformations

In this section, we discuss open issues and our recommendations on transformations from the following aspects: transformation approaches (Sect. 6.3.1), traceability support (Sect. 6.3.2), transformation algorithm (Sect. 6.3.3), and the transformation quality characteristics (Sect. 6.3.4) such as efficiency and scalability.

6.3.1 Approach

As discussed in Sect. 3.2.5, four types of transformation approaches are applied in the primary studies we have reviewed. Selecting which transformation approach to apply is closely related to the representations of source and target models, the complexity and scalability of transformations, and the extent of automation which is targeted.

A classification of transformation approaches is reported in [12, 13], along with a high-level discussion on pros and cons of each type of transformation approaches. For rule-based and pattern-based transformation approaches, as indicated in [12, 13], transformation rules[5] should clearly specify, for example, their application domains, parameters, application constraints, and directions. None of the primary studies of this review clearly specify their transformation rules according to these aspects.

There exist techniques in academia and commercial tools that can facilitate the specification and execution of transformation rules. The Atlas Transformation Language (ATL) [7, 26] is one such model to model transformation technique, developed on top of the Eclipse platform [15], to facilitate the specification, structuring (by packaging rules into modules), and execution of transformation rules. Besides, it provides both declarative and imperative constructs to define transformation rules. However, during the execution of an ATL transformation, its target model cannot be navigated. This often results in complex transformation rules since results from previously executed rules cannot be used as inputs of other rules. Kermeta [27] is an imperative metamodeling language, also built on top of the Eclipse platform, which can facilitate the manipulation of both source and target model elements. Kermeta also supports packages, inheritance, classes, and operations so that transformation rules can be well organized. In addition, another interesting characteristic is design-by-contract for rules: operations implementing rules support pre and post conditions and classes use invariants. There are other academic and commercial tools and languages which can support model to model transformations, such as the IBM Model Transformation Framework (MTF) [23] and the Query/View/Transformation (QVT) standard [39]. A quite exhaustive list of such tools and languages can be found in [13]. We suggest utilizing an existing transformation framework to support transformation from requirements to an analysis model. However, requirements are usually textual, not models. Therefore, we suggest that requirements are transformed into an intermediate model, which can then be further transformed into an analysis model by applying one of the model-to-model transformation techniques.

Another open issue we identified in this review is that many of the approaches do not address the extent to which their generated analysis models are correct and precise enough. One possible evaluation method could be experimentally comparing the analysis model generated by the transformation approaches with the one manually developed by software developers. Research (e.g., [46]) has also been conducted to systematically test and thus validate transformation approaches themselves to ensure that they have the desired behavior. If possible, these approaches should be applied in our context.

6.3.2 Traceability support

Most of the approaches do not address traceability. This is perhaps because in order to support traceability, a mechanism should be proposed to establish and maintain explicit

[5] In [12, 13], patterns are considered as one type of transformation rules.

traceability links between the source and target of each transformation. In cases where multiple transformation steps are involved, traceability links should also be derived for requirements and analysis models from the established traceability links during each transformation step. A traceability link should at least contain references to the source and target elements connected by the link and should preferably indicate which transformation rule(s) are applied to trigger the creation of the link. Another interesting aspect, which is not addressed in any of the approaches, is that transformation rules may rely on the results of other transformation rules, more specifically transformation rules may rely on traceability links established by other transformation rules. One advantage is that transformation rules can thus be simplified. For example, instead of conducting analyses already performed by other rules, we can simply use traceability links. For example, suppose that a class has been generated from a requirement construct (e.g., a noun) and a traceability link has been established accordingly between the generated class and the requirement construct. If a new transformation rule identifies that this noun (requirement construct) is qualified by an adjective (another requirement construct), then the established traceability link can be used to create an attribute in the generated class. This way, the output (traceability link) of the first transformation rule is an input to the second. This mechanism is very useful for transformation approaches that need to query previously generated target elements and trace back to their corresponding source elements through the traceability links previously established.

6.3.3 Algorithm

Not all approaches do provide transformation algorithms. And when they do, they often do not describe their algorithms at a proper level of details that is amenable to an implementation. To facilitate automated transformation, an algorithm should be clearly specified, for example to describe how and when to apply transformation rules. A transformation algorithm should specify the sequence of applying transformation rules, when there are sequential constraints among them. For example, to generate sequence diagrams, one must identify objects before identifying messages exchanged between these objects. A transformation algorithm should also specify how to verify conditions triggering transformation rules. A well-designed algorithm should be easily modifiable when additional transformation rules are added or existing rules are modified. It is possible not to rely on transformation rules (i.e., a transformation is fully described in an algorithm); however, this strategy just works for very simple transformations, which is rarely the case. In cases with a large number of rules, the logic of the algorithm will become very complex and modifications increasingly more difficult. We suggest clearly separating transformation rules from transformation algorithms applying them.

6.3.4 Quality characteristics

Ideally, we also expect transformation approaches to address quality characteristics such as efficiency, scalability, extensibility, and interoperability. Fine-grained transformation from requirements to an analysis model could be very complex; therefore, efficiency and scalability of transformation approaches could become an issue for large software systems. Large-scale case studies are required to evaluate these two quality characteristics. In addition, we also suggest using a minimum number of intermediate models since additional intermediate models unavoidably make transformation approaches less efficient: the more intermediate models, the more difficult the validation and verification of the approach and the higher the chances of loosing information during transformations. In terms of extensibility, it is important to be able to add new transformation rules easily and modify transformation algorithms without too many side effects. It is also desirable that a proposed transformation approach be easily integrated with other approaches or tools, used within a software engineering process such as approaches transforming analysis models into design models, and code generation.

7 Conclusion

In the context of model-driven development, the early step of transforming requirements into an analysis model is a crucial but difficult step. Although mostly performed manually, there have been attempts to automate this software development step. However, despite a significant amount of research, we still do not have a practical, workable automated solution. To gain a precise and structured understanding of the state of the art and identify directions for future research, this paper provides a systematic review of existing work on automating this step. This review systematically selected, investigated, and compared 16 approaches for transforming requirements into an analysis model.

In order to facilitate the synthesis and comparison of the approaches in a systematic manner, a conceptual framework was designed to provide common concepts and terminology for the comparison and evaluation of transformation technologies. This framework also includes a description of the general steps of transforming requirements into an analysis model while establishing traceability links. A set of evaluation criteria, which are derived from the conceptual framework, is proposed to assess each approach in a precise and structured manner. These

evaluation criteria can be adapted to evaluate future research works on the same topic.

Based on the systematic review results, we observed that no existing approach (i) requires acceptable user effort to document requirements, (ii) is efficiency enough (e.g., one or two transformation steps), (iii) is able to (semi-) automatically generate a complete (i.e., static and dynamic aspects), consistent analysis model, which is expected to model both the structure and behavior of the system at a logical level of abstraction, e.g., UML models that at least contain consistent class and interaction diagrams. However, by carefully analyzing and evaluating each aspect of the reviewed approaches, we can make recommendations for future work and a desirable approach can be outlined. A desirable approach is one that can automatically and efficiently transform a use case model using reasonable restrictions to natural language, with or without domain-specific information provided in a glossary, into a complete, correct and consistent UML model comprising both structural and behavioral aspects using one intermediate model and fully automatable requirements preprocessing techniques.

Additionally, our review results show that four types of transformation approaches are applied in the reviewed approaches and selecting which transformation approaches to apply is closely related to multiple factors such as the representation of requirements and analysis models. Existing model to model transformation techniques (e.g., ATL [7] and Kermeta [27]) can be adopted to implement a requirements-to-analysis model transformation approach. Transformation rules and algorithms should be clearly structured and specified. We also summarize and classify transformation rules applied in the reviewed works for future research reference. Our review results also show that most of the approaches do not address traceability. We suggest that a traceability mechanism should be proposed to create and maintain traceability links between requirements elements and analysis model elements. In cases where intermediate models are used, traceability links should also be derived all the way from requirements, through the intermediate models, to the analysis model. Last, we also suggest that research on transformation approaches address, in part through empirical studies, their quality characteristics such as usability, efficiency, scalability, extensibility, and interoperability.

References

1. Parsing, http://www.en.wikipedia.org/wiki/Parsing (Last accessed April 2008)
2. Sentence, http://www.en.wikipedia.org/wiki/Sentence (Last accessed April 2008)
3. IEEE Std. 830-1998, IEEE Standard for Software Requirement Specification, 1998
4. Abbott RJ (1983) Program design by informal English descriptions. Com ACM 26(11):882–894
5. Aizenbud-Reshef N, Nolan BT, Shaham-Gafni Y (2006) Model traceability. IBM Syst J 45(3):515
6. Ambriola V, Gervasi V (2006) On the systematic analysis of natural language requirements with CIRCE. Autom Softw Eng 13(1):107–167
7. Altas Transformation Language (2008) http://www.eclipse.org/m2m/atl/. Last Accessed March 2008
8. Borgo S, Gangemi A, Guarino N, Masolo C, Oltramari A WonderWeb Deliverable D15 Ontology RoadMap, http://www.wonderweb.semanticweb.org/deliverables/documents/D15.pdf
9. Bruegge B, Dutoit AH (2004) Object-oriented software engineering using UML, patterns, and Java, 2nd edn. Prentice Hall
10. Capuchino AM, Juristo N, Van de Riet RP (2000) Formal justification in object-oriented modelling: a linguistic approach. Data Knowl Eng 33(1):25–47
11. Christiansen H, Have CT, Tveitane K (2007) From use cases to UML class diagram using logic grammars and constraints. In: Proceedings of recent advances in natural language processing, pp 128–132
12. Czarnecki K, Helsen S (2003) Classification of model transformation approaches. In: Proceedings of OOPSLA workshop on generative techniques in the context of the MDA
13. Czarnecki K, Helsen S (2006) Feature-based survey of model transformation approaches. IBM Syst J 45(3):621–645
14. Diaz I, Pastor O, Matteo A (2005) Modeling interactions using role-driven patterns. In: Proceedings of IEEE international conference on requirements engineering, pp 209–220
15. Eclipse Foundation, Eclipse Technology Project, http://www.eclipse.org/technology/index.php (Last accessed November 2008)
16. Eclipse Foundation, UML2: EMF-Based UML 2.0 Metamodel Implementation, http://www.eclipse.org/uml2/ (Last accessed November 2008)
17. Feijs LMG (2000) Natural language and message sequence chart representation of use cases. Inf Softw Technol 42(9):633–647
18. Fliedl G, Kop C, Mayr HC, Salbrechter A, Vöhringer J, Weber G, Winkler C (2007) Deriving static and dynamic concepts from software requirements using sophisticated tagging. Data Knowl Eng 61(3):433–448
19. Fliedl G, Mayerthaler W, Winkler C, Kop C, Mayr HC (1999) Enhancing requirements engineering by natural language based conceptual predesign. In: Proceedings IEEE international conference on systems, man, and cybernetics, 5, pp 778–783
20. Grubacher P, Egyed A, Medvidovic N (2004) Reconciling software requirements and architectures with intermediate models. Softw Syst Model 3(3):235–253
21. Harmain HM, Gaizauskas R (2003) CM-Builder: a natural language-based CASE tool for object-oriented analysis. Autom Softw Eng 10(2):157–181
22. IBM Rational Software Architect, http://www-01.ibm.com/software/awdtools/architect/swarchitect/ (Last accessed March 2009)
23. IBM Model Transformation Framework, IBM, http://www.alphaworks.ibm.com/tech/mtf (Last accessed November 2008)
24. Ilieva MG, Ormandjieva O (2006) Models derived from automatically analyzed textual user requirements. In: Proceedings on software engineering research, management and applications
25. Insfrán E, Pastor O, Wieringa R (2002) Requirements engineering-based conceptual modelling. Requir Eng 7(2):61–72
26. Jouault F, Kurtev I (2006) Transforming models with ATL. Lecture notes in computer science, vol 3844, pp 128

27. Kermeta, Kermeta metaprogramming environment, http://www.kermeta.org/ (Last accessed August 2009)
28. Kitchenham BA (2007) Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report EBSE-2007-001
29. Kleppe A, Warmer J, Bast W (2003) MDA explained—the model driven architecture: practice and promise. Addison-Wesley, Boston
30. Kruchten P (2003) The rational unified process: an introduction. Addison-Wesley, Reading
31. Lamsweerde AV (2009) Requirements engineering: from systems goals to UML models to software specifications. Wiley, New Jersey
32. Larman C (2004) Applying UML and patterns, 3rd edn. Prentice-Hall, New Jersey
33. Lethbridge TC, Laganiere R (2001) Object-oriented software engineering: practical software development using UML and Java. McGraw-Hill Education, Boston
34. Liu D (2003) Automating transition from use cases to class model, Thesis, University of Calgary, Department of Electrical and Computer Engineering
35. Mayr HC, Kop C (2002) A user centered approach to requirements modeling. LNI 12:75–86
36. Mich L (1996) NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Nat Lang Eng 2(02):161–187
37. OMG "OCL 2.0 Specification" (2003) Object Management Group, Final Adopted Specification ptc/03-10-14
38. OMG "UML 2.0 Superstructure Specification" (2005) Object Management Group, http://www.omg.org/spec/UML/2.0/
39. OMG "MOF Query/Views/Transformations V1.0" (2008) Object Management Group, http://www.omg.org/spec/QVT/1.0/
40. Overmyer SP, Benoit L, Owen R (2001) Conceptual modeling through linguistic analysis using LIDA. In: *ICSE'01*, 2001, pp 401–410
41. Pender T (2003) UML Bible. Wiley, New Jersey
42. Pressman RS (2005) Software engineering: a practitioner's approach, 6th edn. McGraw-Hill, UK
43. Rilling J, Charland P, Witte R (2007) Traceability in software engineering—past, present and future. CASCON Workshop, IBM Technical Report
44. Salbrechter A, Mayr HC, Kop C (2004) Mapping pre-designed business process models to UML. In: Hamza MH (Hrsg.) Proceedings on IASTED international conference on software engineering and applications. Cambrigde, USA
45. Samarasinghe N, Somé S (2005) Generating a domain model from a use case model. In: Proceedings on intelligent and adaptive systems and software engineering
46. Sen S, Baudry B, Mottu JM (2008) On combining multi-formalism knowledge to select models for model transformation testing. In: Proceedings of ICST 2008, pp 328–337
47. Śmiałek M, Bojarski J, Nowakowski W, Ambroziewicz A, Straszak T (2007) Complementary use case scenario representations based on domain vocabularies. In: Proceedings on MoDELS
48. Somé SS (2003) An approach for the synthesis of State transition graphs from use cases. Proc Softw Eng Res Pract 1:456–462
49. Somé SS (2006) Supporting use case based requirements engineering. Inf Softw Technol 48(1):43–58
50. Sommerville I (2004) Software engineering, 7th edn. Addison Wesley, Boston
51. Subramaniam K, Far BH, Eberlein A (2004) Automating the transition from stakeholders' requests to use cases in OOAD. Proc Can Conf Elect Comput Eng 1:0515–0518
52. Subramaniam K, Liu D, Far BH, Eberlein A (2004) UCDA: Use case driven development assistant tool for class model generation. In: Proceedings of SEKE'04
53. Tan HBK, Yang Y, Bian L (2006) Systematic transformation of functional analysis model into OO design and implementation. IEEE TSE 32(2):111–135
54. Wahono RS, Far BH (2002) A framework for object identification and refinement process in object-oriented analysis and design. In: Proceedings of cognitive informatics, pp 351–360
55. Young RR (2001) Effective Requirements Practices. Addison-Wesley, Boston
56. Yue T, Briand LC, Labiche Y (2009) A use case modeling approach to facilitate the transition towards analysis models: concepts and empirical evaluation. In: Proceedings of MODELS2009
57. Yue T, Briand LC, Labiche Y (2009) A systematic review of transformation methodologies between user requirements and analysis models. Carleton University, Technical Report SCE-09-03