Information and Software Technology 51 (2009) 1291–1307

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

On the generation of requirements specifications from software engineering models: A systematic literature review

Joaquín Nicolás [*], Ambrosio Toval

Software Engineering Research Group, Departamento de Informática y Sistemas, Universidad de Murcia, Campus de Espinardo, 30071 Murcia, Spain

ARTICLE INFO

Article history:
Received 29 October 2008
Received in revised form 27 March 2009
Accepted 1 April 2009
Available online 9 April 2009

Keywords:
Specification generation from software engineering model
Textual requirements generation from software engineering model
Requirements document generation from software engineering model
Systematic literature review

ABSTRACT

System and software requirements documents play a crucial role in software engineering in that they must both communicate requirements to clients in an understandable manner and define requirements in precise detail for system developers. The benefits of both lists of textual requirements (usually written in natural language) and software engineering models (usually specified in graphical form) can be brought together by combining the two approaches in the specification of system and software requirements documents. If, moreover, textual requirements are generated from models in an automatic or closely monitored form, the effort of specifying those requirements is reduced and the completeness of the specification and the management of the requirements traceability are improved. This paper presents a systematic review of the literature related to the generation of textual requirements specifications from software engineering models.

Contents

* Corresponding author. Tel.: +34 968 39 85 25; fax: +34 968 36 41 51.
E-mail addresses: jnr@um.es (J. Nicolás), atoval@um.es (A. Toval).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.04.001

1. Introduction

In a study on future research directions in requirements engineering, Cheng and Atlee [18] affirm that there has been little work on how to interconnect various types of requirements models, and that further research is needed on how to integrate requirements engineering techniques so that they can be used synergistically. According to Goldsmith [29], despite widely held beliefs stating that models and/or code prototypes are the appropriate means to capture and communicate requirements, requirements need to be written in words if they are to be appropriately reviewable. Furthermore, Davis' reflection [20] on the improvement of the requirements management process concludes that the combined use of software engineering models and lists of requirements in natural language is a good practice to improve such a process, since it permits the benefits of the two approaches to be combined. On the one hand, modelling techniques are usually expressive, precise, and facilitate the development team's specification and understanding of the requirements. On the other hand, lists of requirements in natural language can serve as a contract between clients and developers and simplify requirements management. These requirements lists both make the validation of requirements by clients easier and clarify the size of the project and the actual state of the requirements development. The study of the proposals to combine business (system) or software models and textual requirements, thereby, becomes relevant.

The norm in software engineering has been the specification of system or software models from requirements written in natural language. However, if the possibility of extracting information from models as textual requirements in an automatic or closely monitored manner is also included, then the combined use of models and textual requirements is made easier, and some additional benefits can be obtained:

- The effort of writing the requirements is reduced. As is widely accepted (see for example the IEEE 830-1998 standard [31]), a requirement must be unambiguous, complete, consistent and verifiable. The writing of a requirements specification satisfying these criteria is a meticulous task and may require considerable effort. The automatic or closely monitored derivation of textual requirements from models can lead to productivity gains in the requirements specification process.
- The completeness of the requirements specification is improved. The generation of part of the requirements in an automatic or closely monitored manner contributes to the completeness of the requirements specification, since it is easier for the stakeholders to accept or refine requirements than to recall them. In the opinion of Maiden et al. [45], people are better at identifying errors of commission rather than those of omission.
- The traceability between models and textual requirements is automated. CMMI (*Capability Maturity Model Integration*) [16] recommends bidirectional traceability between requirements and development products. The CMMI's Level 2 (within the *Requirements Management* process area) specifically recommends the maintenance of "bidirectional traceability (among requirements and work products)" (SP 1.4).

The overall objective of this work is to carry out a comprehensive review and synthesis of the current research and practices reported in the literature related to the combination of software engineering models and textual requirements in the requirements specification process, and in particular those approaches that generate text with which to document the models. Our intention is to study the literature to find methods and techniques dealing with the generation, translation, combination, integration, or synchronization of (system or software) models and textual requirements, in this order (from models to requirements), thus taking advantage of the benefits listed above. The reverse problem (from textual requirements to models) is not within the scope of this paper. It is a problem that practitioners have traditionally addressed in an informal, *ad hoc* manner, and more research is needed to achieve software engineering models generation which starts from informal requirements.

The objective of this work has been achieved through a systematic literature review (SLR), following the approach of Biolchini et al. [12]. A SLR is a research technique to analyze the state-of-the-art in a particular field of knowledge by formally defining the problem statement, the sources of information, the search strings, the criteria for inclusion and exclusion of the papers found in the searches, the quantitative analysis to be undertaken (if necessary), and the templates for ordering the information collected from the papers. This technique comes from Medical Research and has recently been adapted to software engineering (see the work of Kitchenham et al., e.g. [36,37]; the presentation of the SLR in this paper is particularly inspired by the structure of [37]).

The structure of this paper is as follows: Section 2 summarizes the main aspects of the design of the SLR. Section 3 presents the results of the searches, reports on some deviations from the previously established protocol, and tabulates a synthesis of the results of the SLR which uses a taxonomy to classify the studies reviewed. Section 4 discusses the results based on the research questions formulated in Section 2: the interest in the integration between models and textual requirements is first justified (Section 4.1), and each proposal selected in the SLR is then briefly analyzed (Section 4.2). Product requirements derivation from software product lines models is an interesting issue not covered by the papers found in the SLR. We have therefore decided to perform a kind of "mini-SLR" on this topic which is fully reported in Section 5. Finally, Section 6 presents our conclusions and further work, including five issues that we consider to be key for a RMDB (*Requirements Management Database*) tool supporting a seamless integration between models and textual requirements.

2. Planning the systematic literature review

The review protocol followed in this work is summarized in this section. Section 2.1 states the scope of this research. Section 2.2 presents the research questions that guide the SLR. Section 2.3 presents the planning of the search process. Section 2.4 presents the inclusion and exclusion criteria. Section 2.5 shows the data collected from the selected studies, and finally Section 2.6 shows the data analysis. The SLR protocol has been designed and executed by the first author of this paper as part of his PhD work, and his advisor (the second author of this paper) has revised the protocol, the included and excluded papers, and has discussed the results of the review with him.

2.1. Scope

The overall objective of the SLR has been stated in Section 1, but its scope requires further refinement if it is to be precisely defined. We are interested in reviewing the correspondences m2rs: MOD → REQ and m2rd: MOD → REQDOC that are present in the literature, where:

- *MOD* is the set of all the *models* used in software development that provide a graphical, diagrammatical representation of the specification of an aspect of a system or item of software (e.g. a UML class diagram, an i^* SD context model, a feature model, or even a GUI navigation model). A model is constructed by means of a *technique*. In terms of the Meta Object Facility (MOF) layered metadata architecture [4], MOD consists of all the Level M1 models that are suitable for graphical representation.
- *REQ* is a set that consists of all the sets of *textual requirements*. In short a *requirement* is a condition or capability that must be met or possessed by a system or item of software. A textual requirement is a requirement defined either (1) as a textual statement; (2) as a textual statement plus attributes, including traceability relationships (e.g. IEEE 830 [31]); or (3) as a template filled in with text (e.g. VOLERE [52]). Options (2) and (3) are clearly equivalent. Textual requirements can be specified either by means of natural language (e.g. traditional *shall-statements*) or by a formal language (e.g. SCR [21]).
- *REQDOC* is the set of all system and software *requirements documents*, which usually combine narrative descriptions, textual requirements and graphical models.
- *m2rs* is a correspondence that maps a system or software model *m* to a set of requirements *rs*.
- *m2rd* is a correspondence that maps a system or software model *m* to a requirements document *rd*.

As regards the REQ set, as was explained in Section 1, we are especially interested in textual requirements written in natural language, but formal languages also play an important role in certain specific domains and are therefore included within the scope of the SLR.

Regarding m2rs and m2rd, we are especially interested in those approaches that are suitable for automation, and which allow these mappings to be performed automatically or in a closely monitored form. Those approaches that describe the creation of a requirements document starting from models are also included within the scope of the SLR, despite the fact that they are not presented with the aim of being automated, but solely to "guide" the work of a requirements engineer when writing a requirements document starting from models. Note that m2rs and m2rd are defined as correspondences and not as applications, so that an element in the MOD domain may have more than one image.

2.2. Research questions

The research questions that we intend to answer in this SLR are the following:

RQ1. What value can be drawn from the literature with regard to the generation of requirements specifications (textual requirements and requirements documents) from software engineering models?

RQ2. What techniques have been addressed in this field? (i.e. the techniques used to build the initial software engineering models, the techniques used to build the corresponding textual requirements and requirements documents, and the transformation procedures).

2.3. Search process

The following sources have been selected to perform the SLR:

- IEEE Digital Library (www.computer.org/portal/site/csdl/index.jsp)
- ACM Digital Library (portal.acm.org)
- Science@Direct (www.sciencedirect.com)
- MetaPress (Kluwer + Springer) (www.metapress.com)
- Wiley InterScience (www.interscience.wiley.com)
- Google Scholar (scholar.google.com)

The main journals and events of the software engineering community were sought starting from these sources, in particular those concerning requirements engineering (including conferences and workshops such as RE, ICRE, REFSQ, SREIS, AWRE and WER). Google scholar was selected to complete the set of conferences and workshops searched, and to seek grey literature in the field (e.g. white papers and technical reports).

All of the previously-mentioned sources have search engines based on keywords. The search string defined is the following: ("from" OR "generation" OR "generating" OR "combination" OR "combining" OR "derivation" OR "deriving" OR "integration" OR "integrating") AND ("models" OR "specifications" OR "scenarios" OR "use cases" OR "features" OR "stories") AND ("documentation" OR "documents" OR "requirements").

2.4. Inclusion and exclusion criteria

A tentative application of the search string has shown that, in many cases, it is sufficient to read the title of the contributions to consider them as candidates for selection in the SLR, since the terms of the query are commonly used in literature, and lead to many papers which are not related to the subject of this SLR. When the title is not sufficient to determine the inclusion of the paper as a candidate, the abstract is then read and, if necessary, the introduction and even the whole paper.

With regard to the exclusion criteria, candidate papers presenting RMDB tools which do not present specific procedures for combining models and requirements are not within the scope of this research. Our intention is that this SLR should concentrate upon techniques and transformation procedures, leaving aside the tool market, since (1) it is difficult to obtain access to all the tools (many are proprietary tools); (2) there is a vast number of RMDB tools; and (3) the market changes continuously. Duplicate reports of the same study are also excluded in the SLR: only the most complete version of the study is included. Papers are not excluded on the basis of their publication date (1) in order to be able to detect whether the subject of the SLR was actively addressed in literature during a certain period and then abandoned; and (2) to enable us to discover old papers that could provide ideas which could be adapted to current software engineering techniques.

2.5. Quality assessment

In this SLR the quality of the selected studies is addressed by using the following criteria as a basis:

- *Publication place*. In this respect all the selected sources seek scholarship journals and conferences only, with the exception of Google Scholar, which seeks a wider spectrum of papers. We do not expect a large number of papers in this SLR and thus we are initially open to analyze ideas coming from any journal or conference, including grey literature. We opt to remark in the analysis of the studies which are technical reports or white papers.

- *Tool support*. If any, we study the tool supporting the approach and whether it consists of a prototype or a more mature tool.
- *Validation procedures*. We encode each study with three levels of validation. From lower to higher: (ACS) The study is shown through academic case studies or even through examples. In some cases these are drawn from literature; (ICS) The study has been put into practice in an industrial case study; (IP) The study can be considered as part of the industrial practice in a company or domain, for example because a commercial tool supporting the approach is available in the marketplace. The research method – e.g. Action-Research – , if any, it is also reported.

2.6. Data collection

A data extraction form adapted from Biolchini et al. [12] was filled in for each selected work (see Table 1). The form consists of a section of objective results which correspond to those written by the authors of the work and another section of subjective results related to the reviewers' impressions with regard to the topic of this SLR. The section of objective results brings together the research method of the study (if reported), the problems and limitations reported by the authors (if any), and a summary of the results of the study. When analyzing the contributions, special attention was paid in the results summary to the sentences that reinforce the interest of the subject under study (research question RQ1); the method proposed in the approach, the initial models, and the target statements (research question RQ2); and the RMDB tools involved, together with the validation procedures.

2.7. Data analysis

The data collected were tabulated to show:

- The identifier assigned to the study in the SLR, its authors, bibliographic reference, source and year of publication.
- The classification of the study following the taxonomy proposed (presented in Section 3.2).
- The initial model and the kind of textual statements generated, either in formal or natural language (concerning RQ2).
- The method in which the proposal takes place, tool support, and the validation procedures (concerning RQ2).

In our view, the tabulation of results related to research question RQ1 is not the best manner of presentation since the rationale is of a narrative nature. The justifications for interest in the subject of this SLR that have been collected from the selected studies and which we consider to be of most note are therefore presented in Section 4.1.

3. Results

3.1. Search results and deviations from protocol

The search string was adapted to be used in the search engine of each source. The number of papers found per source is summarized in Table 2, together with those marked as candidates and those finally selected. The search string was formulated by using words in common usage and, after applying the inclusion criteria, most of the studies found were not labelled as candidate studies. After applying the exclusion criteria to the candidate studies, the selected studies were then defined by source. Table 3 shows the candidate studies which were not selected, and why. A total of 23 studies were selected (see Table 2). Finally, identical studies found in several sources had to be removed, resulting in 15 different selected studies.

After studying the bibliographies of the selected papers we noticed an important trend related to the research questions, *literate modelling* (Sections 4.1 and 4.2.1), and believed that the SLR would not be complete if this trend were not reported. We therefore decided to introduce a deviation from protocol by completing the set of selected studies with the sections of bibliography and related work of these 15 selected papers. Three interesting papers not initially found in the SLR were thus selected, which were traced from the bibliographies and related work of the studies that already appeared in the SLR. Six papers that we knew were related to this topic were also selected, despite their not appearing in the searches. We are conscious that this is another deviation from protocol that concerns the repeatability of the SLR, but the interest of working with a more complete set of papers finally prevailed. Twenty-four contributions were therefore eventually selected.

Table 2
Number of found, candidate and selected studies, by source. Identical studies in different sources have not yet been eliminated.

Source	Studies found	Candidate studies	Selected studies
IEEE Digital Library	50	2	2
ACM Digital Library	214	8	7
Science@Direct	45	2	2
Meta-Press	87	2	2
Wiley Interscience	5	0	0
Google Scholar	394	12	10
Total	795	26	23

Table 1
Data collection form.

Objective results extraction	
Study identification	Full bibliographical reference
Study origin	The source/s from which the study has been selected, the paper from which the study is drawn, or (prev.SLR) if the paper belongs to the baseline of papers previous to the SLR
Validation and study methodology	The validation procedures and the research method(s) used to achieve the results, if reported
Study results	The results of the study related to the research questions of the review, including the rationale on which the authors based their work, the initial models, the target requirements, the RMDB tool, and the method used in the proposal, if reported
Study problems and limitations	Those problems and limitations reported by the study authors
Subjective results extraction	
General impressions and abstractions	Here the reviewers raise their own conclusions after reading the study

Table 3
Candidate studies not selected.

Source	Study reference	Reason for rejection
ACM	A. van Lamsweerde, Requirements engineering in the year 00: a research perspective, in: 22nd Intl. Conf. on Software Eng. (ICSE'00), ACM Press, Limerick, Ireland, 2000.	[59] (S8 in Table 4) is a more recent version of the study
Google Scholar	B. Jiang, Combining Graphical Scenarios with a Requirements Management System, Master Thesis, University of Ottawa, Ottawa, Ontario, Canada, 2005	RMDB tool prototype
Google Scholar	N.A.M. Maiden, S. Manning, S. Jones, J. Greenwood, Towards pattern-based generation of requirements from software model, in: Requirements Engineering: Foundation for Software Quality 2004 (REFQS'04), Riga, Latvia, 2004	[45] (S3 in Table 4) is a more recent and complete version of the study

3.2. Synthesis of the proposals

The proposals selected in this SLR are arranged in Table 4, which shows the summarized data from each selected study. These data are analyzed in this section. A more detailed analysis of each paper can be found in Section 4.2.

A taxonomy is proposed for the proposals selected in this SLR, based on the establishment of two dimensions: *combination mode* and *scope* (columns *C. Mode* and *Scope* in Table 4), our intention being to provide a synthesized vision of the field of knowledge addressed by research questions RQ1 and RQ2.

With regard to the *combination mode*, we have identified what we refer to as *generative* and *integrative* approaches:

- *Combination mode: Generative*. These approaches propose algorithms, rules or patterns to generate textual requirements starting from models. These proposals are presented with or without automatic support and the text generated can be written in natural or formal language:
  - *Generative (natural language).* These approaches generate candidate natural language requirements which must be validated. In this group we should stress the work of Maiden et al. on the generation of requirements from i* models [45] (S3 in Table 4); the research of Meziane et al. [47] (S13), to generate English specifications that paraphrase UML class diagrams; the proposal of Firesmith [28] (S19) which derives textual requirements from use cases, scenarios and user stories; and a proposal by Berenbach [10] (S17) which does not, strictly speaking, generate natural language text but rather a hierarchy of requirements from use case diagrams.
  - *Generative (formal language).* These approaches generate requirements written in a formal notation. Formal methods bring benefits such as the mechanical analysis of a system to check for deadlock and livelock freedom, but the adoption of formal methods in industry is challenged by the cost and complexity involved in the formal specification of the system. Hence some approaches in literature have investigated the derivation of formal specifications from requirements models. In this group we should highlight the work on goals models by van Lamsweerde et al., which addresses the automatic generation of operational requirements described by means of pre- and post-conditions and triggers [40] (S5), and the generation of requirements described by means of the tabular technique SCR [21] (S7). In addition, Cabral and Sampaio [15] (S20) research the generation of operational requirements in CSP process algebra from use case specifications.
- *Combination mode*: *Integrative.* These studies do not provide algorithms, rules or patterns to generate requirements from models, but rather a kind of *open-ended guides* to relate models and textual requirements (or, on occasions, a philosophy of work). The resulting requirements can be written in natural language or in a formal notation. This SLR concentrates particularly upon generative approaches but integrative approaches may be of interest in the generation of requirements and requirements documents and have thus also been included. Examples of this are Arlow's proposal [9] (S1) for literate modelling and Firesmith's vision of what a modern requirements specification should be like [27] (S2).

With regard to the *scope* of the approach, some proposals deal with the generation of single requirements (correspondence *m2rs*, explained in Section 2.1) while others deal with the generation of requirements documents (correspondence *m2rd*, described in Section 2.1). Obviously, this dimension is not disjointed, since the same study can address requirements and requirements documents generation. With regard to this SLR both approaches are of equal interest:

- *Scope: Requirement*. These approaches deal with the generation of requirements or sets of requirements, but do not address the requirements documents in which the requirements should be placed. One example is the approach of Maiden et al. [45] (S3) which deals with deriving requirements from i* SD models.
- *Scope: Documental*. These studies concentrate on the manual, automatic or semi-automatic generation of requirements documents. For example, van Lamsweerde's group [59] (S8) has developed a tool (Objectiver) with which to semi-automatically generate requirements documents structured from a goal specification. The literate modelling trend as a whole can also be included in the documental approach, including the works of Arlow [9] and Firesmith [27] (studies S1 and S2 in Table 4).

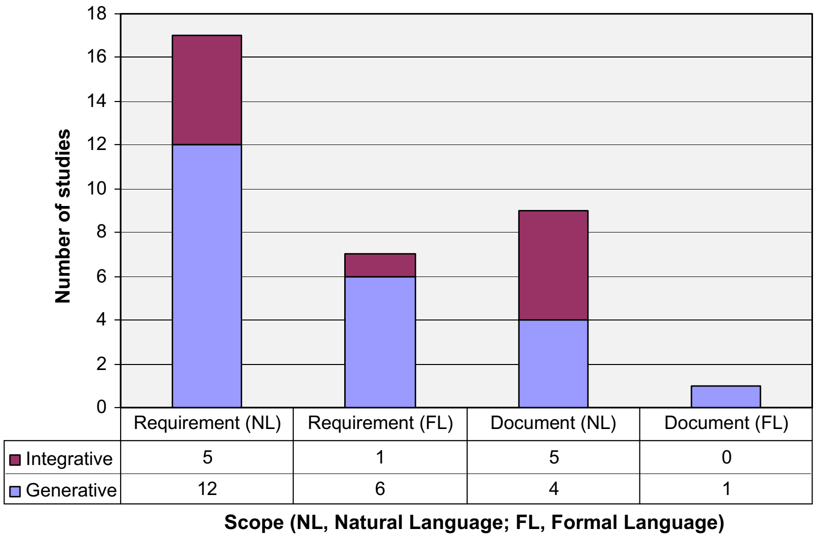
Based on the results in Table 4, Fig. 1 relates combination mode to scope, distinguishing between results in natural and formal language. From this figure we can conclude that more attention has been paid to the generation of single requirements (24 studies) than of documents (10), while generative approaches (23 studies) are more numerous than integrative approaches (11). In relation to the target requirements, i.e. regarding the *Gen.Statem.* column (*Generated Statements*) in Table 4, most of the approaches deal with natural language rather than formal language (26 studies to 8). This column also shows whether the studies report on a particular template for natural language requirements specification (e.g. Volere) or a formal notation (e.g. SCR, KAOS, Albert). Note that the same study may address both requirements and documents and both natural and formal language.

The studies collected in Table 4 are applied to a variety of models (see the *Initial Model* column), which are summarized in Fig. 2. This figure does not compute literate modelling approaches (S1 and S2 in Table 4) because they are applicable to any visual modelling language. The number of studies related to use cases and scenarios (10 studies, 7 of them generative) and goals models (7 studies, 5 of them generative) is particularly noteworthy. On the one hand, although use cases and scenarios have convenient, well-known graphical notations, they are techniques in which text traditionally plays a more important role than diagrams. On the other hand, goal

Table 4
Systematic review studies regarding RQ2.

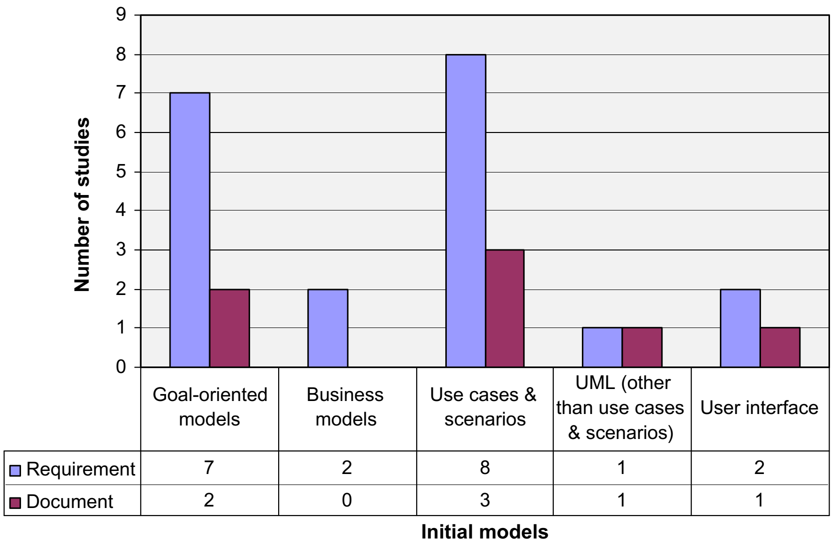
ID	Author/s [ref.] (source)	Date	C. Mode		Scope		Initial Model	Gen. Statem.		Method	Tool	Validation
			Gen.	Int.	Req.	Doc.		FL	NL			
Section 4.2.1. *literate modelling*												
S1	Arlow and Neustadt [9] (very similar and more recent than [8], which is cited by S12)	2004	□	■	■	■	UML or any other visual modelling language	□	■	Not specific	Not reported	(ICS) Enterprise Object Models at British Airways
S2	Firesmith [27] (prev. SLR)	2003	■	□	■	■	Not specific	□	■	Not specific	Not reported	Not reported (vision paper)
Sections 4.2.2–4.2.4. *Goal-oriented requirements frameworks*												
S3	Maiden et al. [45] (acm, mpress, gs)	2005	■	□	■	□	i* goal-oriented model (SD diagram)	□	■ (V)	RESCUE	REDEPEND	(ICS) (A-R) DMAN, air traffic management
S4	van Lamsweerde and Willemet [60] (acm, gs)	1998	■	□	■	□	Scenarios and use cases	■ (K)	□	KAOS	Not included in GRAIL	(ACS) ATM and Lift systems
S5	Letier and van Lamsweerde [40] (cited by S6 and S7)	2002	■	□	■	□	KAOS goal-oriented model	■ (O)	□	KAOS	Validation with StePverif. system	(ACS) Mine pump control system
S6	Alrajeh et al. [6] (acm)	2006	■	□	■	□	Temporal logic goal-oriented model	■ (O)	□	Not specific	Validation with LTSA model checker and Progol5 inductive learning tool	(ACS) Preconditions in KAOS models in the mine pump control system
S7	De Landtsheer et al. [21] (acm, mpress, gs)	2004	■	□	■	□	KAOS goal-oriented model	■ (S)	□	KAOS	Validation through SMV model checker	(ACS) Safety injection system for a nuclear power plant
S8	van Lamsweerde [59] (cited by S3)	2004	■	□	■	■	KAOS goal-oriented model	■ (O)	■	KAOS	Commercial CASE Objectiver (previously GRAIL prototype)	(IP) About 20 industrial projects; Objectiver is a commercial tool
S9	Yu et al. [61] (gs)	1995	□	■	■	□	i* goal-oriented model	■ (A)	□	Not reported	Not reported	(ACS) Banking system
S10	Antón and Potts [7] (prev. SLR)	1998	□	■	■	■	Goal-oriented model	□	■	GBRAM	Not reported	(ICS) (A-R) CommerceNet Web
Section 4.2.5. *Business modelling*												
S11	Cox et al. [17] (sd, gs)	2005	□	■	■	□	RAD business model	□	■	Not specific	Not reported	(ICS) (A-R)e-business system
S12	Türetken et al. [58] (gs)	2004	■	□	■	□	eEPC business model	□	■	Not specific	“KAOS” plug-in for the ARIS toolset	(ICS) two military applications
Section 4.2.6. *UML-based approaches*												
S13	Meziane et al. [47] (prev. SLR)	2007	■	□	■	■	UML class diagram	□	■	Any UML-based method	GeNLangUML Java prototype	(ACS) University system
Section 4.2.7. *Use cases and scenario modelling*												
S14	Maiden et al. [42] (ieee)	1998	■	□	■	□	Scenarios and use cases	□	■	CREWS-SAVRE	CREWS-SAVRE	(ICS) London Ambulance Service
S15	Mavin and Maiden [46] (ieee, gs)	2003	■	□	■	□	Scenarios and use cases	□	■	CREWS-SAVRE	CREWS-SAVRE	(ICS) OCD (naval) & CORA-2 (air traffic managem.)
S16	Maiden and Robertson [44] (acm)	2005	■	□	■	□	Scenarios and use cases	□	■ (V)	RESCUE (evolves CREWS-SAVRE)	ART-SCENE (evolves CREWS-SAVRE)	(ICS) (A-R) DMAN, air traffic management
S17	Berenbach [10] (acm, gs)	2003	■	□	■	□	Use case diagrams	□	■	Any use case driven process	Based on CASE tool scripts	(ICS) Use case models used in Siemens
S18	Berenbach [11] (prev. SLR)	2004	□	■	■	■	Use case diagrams	□	■	Any use case driven process	Not reported	(ICS) Mail sorting system at Siemens
S19	Firesmith [28] (gs)	2004	■	□	■	□	Stories, scenarios and use cases	□	■	Any use case driven process	Not automated	(ACS) ATM example
S20	Cabral and Sampaio [15] (sd, acm)	2008	■	□	■	□	Use case templates	■ (C)	□	Not specific	Ms Word plug-in, CNL/CSP translator, FDR, CSP model checker	(ICS) Research cooperation involving Motorola
S21	Daniels et al. [19] (prev. SLR)	2005	□	■	■	■	Use cases	□	■	Rational Unified Process	Not automated	(ACS) microwave oven SPL example
S22	Probasco and Leffingwell [49] (gs)	1999	□	■	□	■	Use cases	□	■	Rational Unified Process	Rational Suite	Not reported (white paper)
Section 4.2.8. *User interface modelling*												
S23	Jungmayr and Stumpe [33] (gs)	1998	■	□	■	■	Extended usage model	□	■	Not specific	Java prototype	(ACS) UNIrech, bibliographic databases query
S24	Smith [55] (prev. SLR)	1982	■	□	■	□	User-system interface	□	■	Not specific	Prototype on UNIX	Not reported

Source: acm, ieee, sd (ScienceDirect), mpress (MetaPress), wi (Wiley Interscience), gs (Google Scholar), prev.SLR (baseline previous to SLR), cited by [ref.]; *C. Mode* (Combination Mode): Gen.: Generative/Int: Integrative; *Scope*: Req.: Requirement/Doc: Document *Gen.Statem.* (Generated Statements): FL: Formal Language/NL: Natural Language; (K) KAOS, (O) Operation Model, (S) SCR, (A) Albert, (C) CSP, (V) Volere; *Validation*: (ACS) Academic Case Study/(ICS) Industrial Case Study/(IP) Industrial Practice; A-R: Action-Research.



	Requirement (NL)	Requirement (FL)	Document (NL)	Document (FL)
Integrative	5	1	5	0
Generative	12	6	4	1

Fig. 1. Number of studies by scope and combination mode.



	Goal-oriented models	Business models	Use cases & scenarios	UML (other than use cases & scenarios)	User interface
Requirement	7	2	8	1	2
Document	2	0	3	1	1

Fig. 2. Number of studies by scope and initial model.

models benefit from van Lamsweerde et al.'s attention to KAOS (4 generative studies, S4–5 and S7–8 in Table 4). Two approaches concern business modelling, but none of them addresses document generation. There are also two studies on interface models. It is worth noting that there is only one generative approach on UML, leaving apart use cases and scenarios. It seems that more effort should be made in this area, particularly in those UML techniques which are more suitable for use in requirements engineering.

With regard to the *Method* column in Table 4, few studies are part of a software development method. This is hardly surprising since we believe that these approaches can be used in the context of any method in which the initial models are used.

The first quality assessment criteria established in Section 2.5 is *Publication place*. Only one white paper appears in the selected studies, which is that of Probasco and Leffingwell [49] (S22). The remaining studies do appear to have been published after a referee process with, perhaps, the exception of the two columns of Firesmith at JOT [27,28] (S2 and S19). Arlow and Neustadt's book chapter on literate modelling [9] (S1) comes from a contribution to a refereed conference [8]. Data regarding *Tool* and *Validation* can also be found in Table 4:

- As regards *Tool*, to the best of our knowledge, REDEPEND and Objectiver are the most mature tools found in this SLR. Objectiver is even commercially available. Besides, automation is not reported in some of the studies, notably in Arlow and Neustadt's proposal [9] (S1) for literate modelling and in Firesmith's *vision* of modern requirements specification [27] (S2). The remaining approaches which report automation show their viability by means of prototypes whose real maturity level is difficult to assess from the information reported in the papers.
- Fig. 3 presents a summary of *Validation*, and makes it is clear that real industrial practice is scarce in this field. A case study in a company must overcome an important gap if it is to become part of that company's real practice. The reading of the papers does not allow us to make a precise assessment of the size of this gap in all cases. The number of academic and industrial case studies is similar (9 and 11 studies, respectively). Industrial case studies

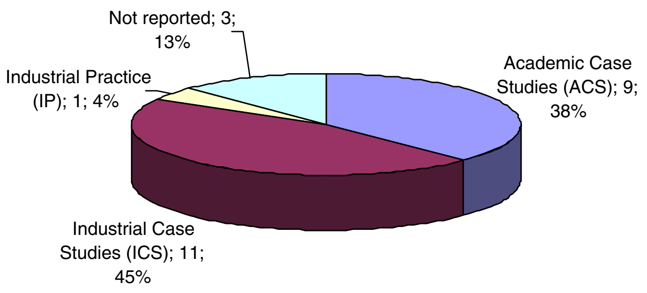


Fig. 3. Types of validation (including number of studies and percentage).

are in principle preferable, although it is often difficult to make a precise assessment of the real scope of an industrial case study by reading the papers. Nevertheless, some papers present rigorous theoretical research and are illustrated by means of an academic case study (e.g. S4, S5 and S7 in Table 4 by van Lamsweerde et al.).

Finally, but of no less importance, no approach exists to address the issue of maintaining synchronization between the documents or requirements generated and the initial models. In the papers selected the generation always takes place in one direction, from models to requirements. No approach exists to permit the changes in the generated requirements to be automatically propagated in the initial models. We believe that this synchronization could be useful in an iterative and incremental software process, especially during validation with customers. Validation could therefore be carried out directly on the widely understandable generated textual requirements, which could be changed to make the related models evolve automatically through traceability relationships. We are obviously referring to the synchronization of certain predefined changes in the generated textual requirements or documents, since synchronization in general may be difficult.

4. Discussion

Having filled in the data collection forms and analyzed the selected contributions, we now present answers to the research questions presented in Section 2.2.

4.1. RQ1. What value can be drawn from the literature with regard to the generation of requirements specifications from software engineering models?

An a priori justification of the interest of this topic was formulated in Section 1, before the SLR was performed. This question (RQ1) permits us to look for statements found in literature which justify an interest in this topic. In this section we collect some of the statements that we consider to be most representative.

Arlow and Neustadt [9] (S1 in Table 4) believe that in many cases not all stakeholders are able to understand the syntax and semantics of a business model that uses UML and other visual models. Furthermore, these authors discuss the following concerns with regard to visual models:

- In order to access the information embedded in a model it may be necessary to know how to operate a modelling tool. The reports that all modelling tools generate – usually in HTML format – are often difficult to read and navigate and are thus, in Arlow and Neustadt's experience, of limited practical use.
- Unless one is familiar with the general "shape" of a visual model, it can be difficult to determine where to start reading, either when reading the model in a modelling tool or when reading a generated report.
- It is sometimes difficult, or even impossible, to uncover the business requirements and the rationale underlying the visual model, as these become *invisible* when taken out of the business context and expressed in a visual notation. For instance, a highly important requirement can be expressed in such a concise way that the requirement can be easily overlooked during a walkthrough. Arlow and Neustadt call this "the trivialisation of business requirements by visual modelling" [9]. The same authors show this *trivialisation* through an example in which a highly important business requirement is finally expressed in the models as an association multiplicity "n" rather than "1": this requirement could easily pass unnoticed during the validation of the model.

Literate modelling is a technique drawn from the *literate programming* proposed by Knuth in the eighties, in which modelling and natural language documentation are seamlessly bound together in a synergetic manner, and the functionality of CASE (*Computer-Aided Software Engineering*) and RMDB tools are truly integrated. We believe that many software engineers have been using literate modelling intuitively in an ad hoc manner throughout their professional careers, without being aware of this term. In a paper referenced by Türetken et al. [58] (S12), Finkelstein and Emmerich [26] examine the future of RMDB tools and place literate modelling within the long-term future of these tools. These authors claim that there is no suitable manner in which to synergistically use models and natural language in RMDB tools. These authors foresee that literate modelling has a role to play in these tools. There is a school of thought that argues that natural language requirements are a vestige of outmoded practice which survives because of the lack of technological transfer to the IT industry of R&D modelling methods. Finkelstein and Emmerich, however, believe that natural language plays a valuable role which is, furthermore, unlikely to be supplanted. Natural language is useful to show the correspondences between the components of the models and those of real-world phenomena, and allows the stakeholders to validate the specification. Meziane et al. [47] (S13) add another benefit: the automatic generation of natural language requirements for maintenance purposes. Software implementations are often not consistent with the documentation, since developers do not update analysis and design models when they change the code. Design models can be generated from the evolved implementation by means of a tool allowing reverse engineering. It would therefore be useful if natural language requirements were generated based on that updated design.

Maiden et al. [45] (S3) believe that their work is part of an important trend towards the integration of requirements models. These authors state that although many model-based specification and analysis approaches with which to specify the requirements of computer-based systems exist, most organizations continue to represent requirements textually. "Unfortunately", they write, "most modelling approaches have not been designed to support the derivation of requirements statements from models, or to be used alongside textual requirements descriptions".

When van Lamsweerde [59] (S8) exposes the lessons learned in the application of goal-oriented requirements engineering, he states that "the diversity of requirements engineering projects in type, size, and focus call for highly flexible technologies. We felt that a *multi-button* method and tool that by default supports graphical and textual specifications, plus formal specifications only when and where needed for incremental analysis of critical model fragments, is a promising step in that direction". van Lamsweerde also asserts that requirements documents are generally perceived as being big, complex, outdated, and too far away from the executable products customers are paying for. He adds that "in the end, what bothers customers the most is the quality of project deliverables. The model-driven requirements document generated with our tool [Objectiver] was perceived as the main success indicator in many projects".

Firesmith [27] (S2) discusses the problems of traditional requirements specifications and proposes the requirements in an approach to manage those problems in an iterative and incremental software process. Such an approach should enable the automatic generation of audience-specific requirements specifications. For example, the needs of executive managers or project directors are very different from those of software testers. With regard to iterative process models, the need for requirements documents to have unresolved questions annotated is made explicit in Antón and Potts's approach [7] (S10) towards goal-oriented requirements engineering. In a requirements specification the listing of requirements can be as important as the tracking of open issues, which appear and disappear dynamically.

Türetken et al. [58] (S12) work on the automatic generation of requirements in natural language from business process models and state that their approach can be used to manage the minimization of "non-value-added tasks" such as the rewriting and documenting of requirements, thus improving their quality as well as facilitating their modification. The functional requirements of the system and the software are generally based on visual models, but the traceability and the interaction between these models and the requirements represented in natural language are generally missed. One of the benefits of their approach is that the standard format for the requirements generated simplifies the understanding, verification, validation, and management of these requirements for all stakeholders.

Berenbach's [10] (S17) experiences in Siemens have led him to observe the disconnection between a UML model and the requirements of the processes modelled. This author believes that this gap tends to widen: as models become more complex the extraction of detailed requirements becomes more difficult. Failure to develop complete requirements sets from UML models may have serious consequences at a later date. For example, the derived test cases might not provide complete coverage. Berenbach [11] (S18) states that the solution arises from the planned integration between models, the text of the use cases, and the requirements. He believes that "a UML model is a repository, not a set of diagrams". He also provides a list of best practices, including the suggestion that documentation should be generated "on demand".

With regard to the integration of use cases and textual requirements, Daniels et al. [19] (S21) state that use cases provide understandability, context, and direct traceability to actor needs and interfaces, while shall-statement requirements add the precision necessary to completely and unambiguously specify the system. Despite the usefulness of use cases, shall-statements, diagrams, tables, equations, graphs, pictures, pseudo-code, state machines and other methods must still be used to capture additional requirements and add richness to provide a sufficient level of detail to characterize a system. These authors conclude that "use case models and traditional shall-statement requirements are synergistic specification techniques that should be employed in a complementary fashion to best communicate and document requirements".

4.2. RQ2. What techniques have been addressed in this field?

For reasons of clarity, the selected proposals described in this section have been grouped as they are shown in Table 4.

4.2.1. Literate modelling

A detailed introduction to the intention of literate modelling was given in Section 4.1. Arlow et al.'s original proposal on this topic [8] is later refined in [9] (S1 in Table 4). In this approach a new type of document is defined, *Business Context Document (BCD)*, in which (1) the diagrams are embedded as figures in a document written in natural language and the visual model is paraphrased (for example, in relation to certain UML associations it can be written as "Each Product has a Price List which contains zero or more Prices"); (2) brief explanations of the relevant UML syntax are included in footnotes; (3) real, yet simple examples of the notation are presented to illustrate specific points; and (4) some important questions such as the rationale are emphasized. Literate models are thus UML models that are embedded in a natural language text which explains them.

In Arlow et al.'s experience, it is preferably to structure BCDs around the key *things* that deliver value to the business rather than the business *processes*. Things are more stable than processes and tend to naturally form cohesive clusters that provide a suitable focus for the BCD. A simple relationship between the package structure of the UML models and the BCDs can be expected. This proposal does not, however, mention specific patterns or algorithms for generating requirements from UML models: instead, the authors describe the base contents of the BCD and provide guidelines with which to create it. According to Arlow et al., each BCD has the following base structure: (1) business context; (2) compliance to standards; (3) a roadmap UML model, showing all the main things and relationships, with cross-references to the appropriate parts of the BCD; and (4) a number of sections, each describing a thing or related things and comprising: (4.1) narrative, referencing one or more model fragments; (4.2) UML diagrams illustrating the narrative (which, according to the authors, are typically class diagrams, use case diagrams, and sequence diagrams); and (4.3) informal diagrams wherever they enhance the description. Arlow et al. also provides guidelines on the writing style to be used in BCDs and, for example, recommend the use of concrete examples to illustrate the models and the development of a business nomenclature.

We believe that, in a broad sense, the proposals collected and analyzed in this SLR could be considered as a part of this trend in literate modelling, although no specific reference is made to this in the papers themselves. We particularly believe that Firesmith's proposal with regard to requirements specifications [27] (S2) could be considered as literate modelling. The goal of this approach is to obtain a correct, complete, consistent, current, and audience-appropriate (i.e., supportive of the role-specific tasks of its numerous audiences) requirements specification. To improve the requirements specification, and based on his experience, Firesmith proposes the use of a tool that (1) has a fine-grained repository; (2) enables automatic specification generation, as a kind of separation between Model and View in the MVC (Model-View-Control) paradigm; and (3) generates different specifications for different readerships. Another interesting idea that Firesmith points out is the need to establish some kind of publish-subscribe mechanism which permits stakeholders to be notified when certain requirements of interest to them change in an iteration or are added in incremental development. This feature would, for example, be critical to a designer.

In conclusion, in the words of Arlow and Neustadt [9]: "in practice, literate modelling, although a very good idea, didn't really take off well. This was partly because of its reliance on special text-processing tools that were not widely available and partly because programmers generally prefer to write code rather than narrative! In contrast to this, literate modelling has proven to be very popular with those that have tried it. This is because a literate model not only provides a context for a UML model that is otherwise lacking but also helps the modeller do his or her work". We believe that if this proposal of literate modelling is to be successful it should be based on the automatic or closely monitored generation of the BCD or significant parts of the BCD. If BCDs have to be created entirely manually, then literate modelling will probably not be applied in practice: the manual generation of BCDs can be cumbersome, and BCDs can be hard to write because, according to Arlow and Neustadt, they require a very sound and broad overview of the business, good UML skills, and good writing and communications skills.

4.2.2. RESCUE and REDEPEND: generation of candidate natural language requirements from the i* framework

In the set of proposals of Neil Maiden et al. regarding the requirements engineering process RESCUE, one of the premises is that it is easier for the stakeholders to identify errors of commission rather than those of omission. In other words, the stakeholders find it easier to read a possible alternative scenario in a use case or a candidate requirement in natural language and to accept or reject it than to recall all the possible scenarios or all the requirements of the system. In this context, two set of papers can be found in the scope of RESCUE: (1) those generating requirements in natural language from i* SD – *Strategic Dependency* – context models (these are basically graphical context models that allow the goals of the interactions between the agents of a socio-technical system to be captured); and (2) those attempting to complete the set of alternative scenarios in the use cases (described in Section 4.2.7).

In the first set of papers, Maiden et al. [45] (S3) describe 19 patterns that have been developed to *paraphrase* an i* SD model thus making information already included in the model explicit. These patterns identify recurring syntactic and semantic structures in the i* models, from which candidate requirements are generated that must be validated. These requirements are specified in natural language by using the VOLERE template. This approach was applied manually in an industrial case study, DMAN (*DEparture MANager*), an air traffic management system. The authors consider this approach to be a success because 214 requirements statements were generated manually in three days, of which 207 were included in the final requirements specification (encompassing almost 900 requirements). Therefore, almost 25% of the requirements in the final specification were generated in this manner. This approach helps to improve the completeness of the requirements specification although, as Regnell et al. [51] point out, probably not the elicitation per se since the information must already be included in the i* model. This approach can be useful in any system in which a set of heterogeneous actors interact in order to achieve certain dependent goals. Maiden et al. conclude this study by asking themselves whether an automatic, pattern-based generation of candidate requirements statements would be cost-effective, or whether the number of duplicate and false-positive requirements would be unacceptable.

An initial answer to this last question raised by study S3 can be found in a subsequent paper, [41], in which Maiden et al. present the lessons learned during the application of i* in several industrial case studies, and discuss the use of a new version of REDEPEND which automates the applications of the previous patterns. For the purpose of this SLR, REDEPEND enables analysts to construct i* SD models during a requirements workshop, to automatically generate candidate requirement statements from this i* model in real time, and to then walkthrough these generated requirements to select and reject them. The 19 previous patterns are represented in an Ms Excel file in REDEPEND, meaning that eventual new patterns can be added without changing the tool. In the previous DMAN case study, the automatic generation of 287 candidate requirements took REDEPEND only 12 s when running on a standard PC. It should be noted that the number of candidate requirements generated is larger than in [45] because the 19 patterns had been refined, thus leading to more requirements of different types. With regard to requirements generation, the initial evaluation of the tool made by Maiden et al. is positive, although they aim to gather and report evaluation data in the future.

4.2.3. Goal-oriented requirements engineering with KAOS and Objectiver

van Lamsweerde et al. have produced an authoritative collection of work around KAOS, a goal-oriented framework that includes method and tool support. The KAOS method has been applied in about 20 industrial projects at CEDITI (a university spin-off) [59], covering a variety of domains. Natural language is used in KAOS to describe the system informally, although when necessary a temporal logic can be used to describe the system formally. This section addresses the generation of both textual requirements and requirements documents. With regard to textual requirements, the requirements generated are not specified in natural language but in formal notations (pre/post-conditions & triggers and SCR). The starting models are scenarios and goal models, which have convenient graphical diagrams, although they are processed from their textual representation.

The first work in this section is related to scenarios, which are widely considered to be an effective means to elicit, validate, and document requirements. However, scenario models are partial and may pass over certain properties of the system. Goals, requirements and assumptions related to scenarios are, in addition, only implicitly described. The system's properties must therefore be explicitly expressed to permit an analysis of consistency and completeness to be carried out. A method was thus developed in the realm of KAOS to systematically infer formal, abstract declarative specifications of goals, requirements and assumptions starting from informal, concrete scenarios [60] (S4). The generated specification uses temporal logic. This proposal is grounded in the experience of the KAOS research group in numerous projects but, according to this paper [60], the putting into practice of this proposal in a real industrial case study is still pending.

After this work, research was then conducted in the realm of KAOS to project declarative goals models to operational requirements in order to achieve the best of both approaches:

- Operations specified by pre- and post-conditions and triggers [40] (S5). The functional goals assigned to the software agents must be operationalized in the specifications of the software services that the agents should provide to meet those goals, by applying *operationalization patterns*. The authors recognize their limited experience with these patterns, mainly based on their handling of a variety of case studies from literature. Alrajeh et al. [6] (S6) affirm that this refinement of operational requirements from goal models is a tedious manual task only partially supported by operationalization patterns. These authors address this problem by proposing a semi-automated approach based on model checking and inductive learning. This is initial research which is described through the case of learning preconditions for KAOS models. Alrajeh et al. believe that this method can be tailored to generate other operational requirements such as triggers.
- Tabular event-based specifications for control software, written in the SCR language [21] (S7). These specifications are a well-established method with which to specify operational requirements for the development of control software, and provide sophisticated techniques and tools for the late analysis of software models. This proposal is again shown through an example found in literature.

In a keynote at RE'04, van Lamsweerde [59] (S8) reflected on goal-oriented requirements engineering research and practice. One of these reflections concerned "the need for effective tool support", and Objectiver was presented, together with some related lessons learned and challenges. Objectiver is a commercial toolset (www.objectiver.com) supporting the KAOS method. This is a mature and well-documented environment which evolved from the GRAIL prototype. The Objectiver toolset makes it possible to generate requirements documents in a closely monitored form. The structure of these documents stems from the goal refinement graph and from requirements documents templates which reflect company-specific standards (IEEE 830 [31] is also included). The requirements document contains a glossary of terms generated from the object model, textual annotations retrieved from the model, and figures selected by the user through drag-and-drop from the goal, object, agent and operation sub-models. Objectiver is the only work found in this SLR that we would be sure of cataloguing as "industrial practice".

4.2.4. Other requirements specification derivations from goal modelling

The joint proposal of the Yu & Mylopoulos and Du Bois & Dubois groups [61] (S9) combines two agent-oriented frameworks for requirements engineering: the specification of (primarily functional) requirements with Albert starting from goal-based business modelling with i*. Albert is a rigorous language which uses a first-order temporal logic. This approach assumes that the requirements process can iterate between the levels of *specification* and *understanding* towards a requirements specification: the level of specification or modelling prescribes *what* agents should do or know, and the understanding or analysis level describes *why* agents relate to each other in a certain way, and why they might prefer some other configuration of relationships. The objective of this work is not to translate semi-formal i* diagrams into Albert formal specifications but to use i* as a first modelling notation for eliciting high-level goals before converting them into finer formal requirements. This is, therefore, an "integrative" approach (see *C.Mode* in Table 4), but is a preliminary, early work which should elaborate on the method needed to obtain the system requirements. The proposal is shown through an academic example.

GBRAM (*Goal-Based Requirements Analysis Method*) is a requirements engineering method outlined by Antón and Potts [7] (S10), and is used to infer goals from espoused requirements and to then derive more complete operational requirements from those goals. This study has been included in the SLR since the last activity of GBRAM related to goal refinement is "Operationalize", which consists of translating goals into operational requirements for the final requirements specification. These requirements are specified by means of templates containing a refined goal, pre and post-conditions, and associated scenarios. Note that the mode of this study is "integrative" and not "generative" (Table 4). GBRAM does not, therefore, propose algorithms, rules or patterns with which to derive the operational requirements, but provides a set of heuristics and involves the timely posing of systematic questions, the relaxation of initial goals by considering *obstacles* (anything that can happen to thwart a goal), and the exploration of scenarios. An interesting contribution of this paper is that requirements documents should be "living documents" in the sense that they keep track of requirements, open issues that appear and disappear dynamically, and *organizational* requirements that encode certain important information for the requirements engineering process (for example, the person who has a good knowledge of certain requirements, or the person who will ultimately be affected by a decision). The output of the whole process is a requirements document whose structure is based on the main functional areas within the system, each containing the following subsections: Goals, Functional Requirements, Non-functional Requirements, and Organizational Requirements. One limitation of the study is that the link between goals and non-functional requirements is not considered. This work, conducted by means of Action-Research, has been validated in several real case studies including an e-commerce application.

4.2.5. Deriving requirements from business modelling

The generation of requirements from business process models includes an article by Cox et al. [17] (S11) which attempts to (1) discover the applicability of Michael Jackson's *problem frame* propositions to complex, industrial projects; (2) link business process models by means of the RAD (*Role-Activity Diagram*) notation on these problem frames; and finally (3) derive requirements from the process models. In our view, this *derivation* of requirements is not systematic, but is rather one of the steps to obtain a problem frame related to the process model and involves the production of a requirements specification in which the requirements engineer applies his or her knowledge to the problem. We believe that, in spite of the title of the paper, the derivation of requirements plays a secondary role in this work. The research is validated through an industrial e-business system and the case study is conducted using Action-Research.

The research into the derivation of requirements starting from business process models also includes the contribution of Türetken et al. [58] (S12), who propose a pattern to specify part of the process models written in eEPC notation in natural language. This pattern is automatically applied through a tool and, according to the authors, provides an important productivity gain, although 40% of the requirements generated needed modification. The reported case studies are two large military applications. This work, therefore, is in line with those of Maiden et al. (Section 4.2.2), although only one requirements pattern is used and it does not take into account the generation of conditional sentences.

4.2.6. Deriving requirements from UML models

Arlow et al.'s proposal for literate modelling (Section 4.2.1) is related to UML but does not provide specific algorithms, rules or patterns to derive requirements from UML models. Apart from use cases and scenarios (discussed in next Section 4.2.7), only one proposal regarding UML has been found.

Meziane et al. [47] (S13) introduce another interesting line of work in the SLR: natural language generation systems. They propose the *GeNLangUML* (*Ge*nerating *N*atural *Lang*uage from *UML*) prototype, which generates English specifications that "paraphrase" UML 1.5 class diagrams by making use of a linguistic ontology called *WordNet*. Meziane et al.'s aim is twofold: (1) on the one hand they wish to provide users with two different views of the system specification at any time: UML and natural language; (2) on the other hand they see the motivation for their tool in a reverse engineering process during the maintenance stage, to enable backwards transformation allowing the stakeholders to "visualize" the changes in the system's implementation in natural language. System evolution is derived from source code, design notation – UML – and system specification in natural language. Text generation is exemplified by means of an academic case study on a University system. The specification generated includes statements such as "A person is a professor or a student", "A professor has a name, a home address, parking privileges, a date, a seminar and seminar overseen", "Zero or many professors instruct zero or many seminars", "A person lives at an address", "An address has a street, a city, a state and a zip code", and so on. GeNLangUML extends the *ModEx (Model Explainer)* system, an earlier work by Lavoie et al. [39].

Meziane et al. identify some weaknesses in their approach, which is purely academic. Firstly, the proposal relies on naming conventions extracted from text books. These conventions might change in industrial practice and the tool would have to be configured accordingly. Secondly, the level of abstraction of the text generated is close to the level of abstraction of the initial UML model. Thirdly, the authors postulate that an ideal system to generate natural language from object-oriented models would include class diagrams, interaction diagrams (sequence and collaboration), state diagrams, activity diagrams and OCL. Regarding interaction diagrams, we doubt that it is necessary to include collaboration diagrams since in our experience they are not concerned with requirements, but with design. Sequence diagrams can, on the contrary, be used to describe use case scenarios. Finally, the authors of this work state that it can be complemented by Burke and Johannisson's [14] research into the translation of OCL specifications into natural language.

4.2.7. Use cases, scenarios and user stories

Use cases constitute a well-known technique to elicit, analyze, specify and validate requirements that have a simple, widespread graphical representation, although use cases are mainly textual in nature [38]. This section deals with use cases, scenarios, and user stories in combination with textual requirements and documents. We have included a heterogeneous set of proposals whose intent is different but, we believe, complementary in that they can be applied by following a logical sequence of action. (1) We first discuss Maiden et al.'s research aimed at improving the completeness of requirements by analysing system scenarios [42,44,46]. This process uses the existing use case model as a starting point and derives new scenarios, taking into account situations which have not yet been considered. This derivation of new scenarios leads, in turn, to new requirements. (2) We then examine the works of Berenbach [10,11] and Firesmith [28], which address the generation of natural language requirements from use case models. Berenbach is concerned with generating the hierarchy of requirements while Firesmith proposes a pattern with which to derive the requirements' text. Cabral and Sampaio [15], in contrast, research the generation of formal language requirements from use case models. (3) Finally, Daniels et al. [19] and Probasco and Leffingwell [49] deal with the requirements documents to be developed from use cases and scenarios, integrating traditional shall-statement requirements and use cases. These last two papers do not strictly deal with the generation of textual requirements from models, but have been included in the SLR in order to exemplify the guidelines concerning the contents of the requirements document to be built when use cases and textual requirements are combined in requirements engineering.

Maiden et al. have carried out long-term research to study the development process of scenarios and use cases to identify missing requirements needed to ensure the requirements specification is complete. These authors have developed a database containing a hierarchy of abnormal and error conditions that is used to generate candidate, alternative event courses to be validated. These alternate courses are raised from *what-if* questions related to a normal course of events. The database contains 54 classes of abnormal behaviors and states, which can be used either manually as a checklist for each event or automatically through the ART-SCENE tool in the RESCUE process. This general taxonomy has been extended with knowledge from several application domains. An early version of this tool (then called CREWS-SAVRE) is presented in greater detail in Maiden et al. [42] (S14) through the example of a retrospective scenario analysis of part of the London Ambulance Service. Mavin and Maiden [46] (S15) then go on to study what types of scenario and which walkthrough techniques are most effective for discovering requirements through two case studies: (1) a simulator in the naval warfare domain at BAE SYSTEMS; and (2) Conflict Resolution Assistant (CORA-2) in the air traffic management system domain at Eurocontrol. Mavin and Maiden provide guidelines for scenario-based requirements discovery, and curiously suggest that systematic walkthroughs of simple scenarios that do not contain excessive domain knowledge are more effective for discovering system requirements. Building on this work, Maiden and Robertson [44] (S16) carry out a retrospective analysis of a previous experience to investigate how use cases and scenarios were developed during the application of the scenario-based RESCUE process in DMAN, an air traffic management system for the UK's National Air Traffic Services. Maiden and Robertson also study the connection between scenarios and requirements in natural language specified through the VOLERE template.

Maiden et al.'s influential research is still being extended. We shall now report on additional, subsequent work with the aim of complementing the analysis of the previously selected studies. This consists of (1) an extension to ART-SCENE to include rich media scenarios [62], which serves to study the improvement of requirements discovery by using other scenario forms, such as visual simulations of agents in the domain, and which are presented to the stakeholders alongside text scenarios in ART-SCENE; and (2) the Mobile Scenario Presenter (MSP) tool [54], which is an extension of ART-SCENE and serves to investigate the use of PDAs (*Personal Digital Assistants*) to undertake ART-SCENE scenario walkthroughs on site. Maiden et al. study then [43] whether visual simulations of scenarios and scenario walkthroughs in the work context can trigger requirements that might not be discovered with ART-SCENE scenario walkthroughs.

Berenbach [10] (S17) has designed an algorithm for the automatic extraction of requirements from use case diagrams. In our view, this author conceives use case diagrams in a particular way, as *conceptual diagrams* or rather as *feature diagrams* [35] of the domain under study. This algorithm is used to create a requirements tree: the so-called *abstract use cases* are mapped onto *features* and *sub-features*, while the so-called *concrete use cases* are transformed into *detailed requirements* for which project tasks may be created and test cases can be generated. This approach concentrates on the generation of the hierarchy of requirements rather than on the requirements text itself. This work has been tested in several complex models from Siemens operating companies. In [11] (S18), Berenbach builds on these results to report on the synthesis of text-based requirements and the so-called *model-driven requirements engineering*. This synthesis results in a requirements engineering approach that seamlessly integrates use cases, features and requirements. In this work Berenbach proposes a set of best practices and recommendations, including automated use case and SRS document generation. Berenbach's industrial experience has led to the discovery that the approach described in this paper [11] takes about one third to one half the time of a traditional approach owing to scalability.

Firesmith's aim [28] (S19) is to derive a set of complete, unambiguous, and verifiable requirements starting from "incomplete and vague" stories, scenarios and use cases. To that end, Firesmith proposes the derivation of textual requirements from use case path interactions by using the following standard format: "If a *trigger* occurs when certain *preconditions* hold, then the system shall perform a required set of *actions* and shall be left in a required set of *post-conditions*". This template is manually applied to the use case specification, producing a requirements specification in natural language which the stakeholders are able to understand and validate. Firesmith postulates that the extra work needed to build a requirements specification is therefore soon recovered owing to the effort saved during the other software development activities. The textual requirements generated are specified in natural language to enable stakeholders to validate them. However, these requirements are long and complex and their readability may be difficult. This problem, which Firesmith associates with the "unavoidable complexity of complete requirements", can be mitigated by breaking the sentences into their constituent parts. The author shows the applicability of the proposal through an academic, classical ATM example.

The research of Cabral and Sampaio [15] (S20) introduces the generation of formal specifications from use case models. These authors propose a strategy through which to automatically translate use cases written in a subset of English (CNL, *Controlled Natural Language*) into a specification in CSP process algebra. CNL includes an ontology which describes the specific entities in the application domain. Use cases are specified by means of two levels of templates: (1) *user view use cases*, which design how actors interact with the system; and (2) *component view use cases*, which specify the system behavior based on user interaction with system components. This approach can be useful in those domains in which requirements consistency is especially important, such as telecommunications. This work specifically arises from a research project in collaboration with Motorola. The resulting formal specification can be used to automatically generate test cases. This formal specification is not legible to stakeholders, but we do not believe that this is a problem since they can validate the initial use case model. The use of a controlled language also helps to avoid ambiguity in use cases templates. This approach is entirely supported by a toolkit consisting of an Ms Word *plug-in* to edit use case specifications according to the CNL grammar; a translator of CNL use cases to CSP; and FDR, a CSP model checker, to check refinement between use and component views.

Daniels et al. [19] (S21) propose a practical method with which to integrate use cases and shall-statement requirements. For these authors use cases are not requirements, but a vehicle to discover requirements. They propose the use of a *Functional Requirements Segment* in the *Specific Requirements* section in the use case template in which to contain the functional shall-statement requirements that the requirements engineer extracts manually from the sentences of the scenario (note that a sentence in a use case can contain multiple functional requirements). The textual requirements retain their context since they are traced to the use case event or sequence of events from which they were derived. When the requirements are documented as a set of shall-statements without context it is difficult to comprehend them and fully interpret their intent and dependencies. The *Supplementary Requirements Specification* also contains the requirements that do not fit well within the context of only one use case. If a traditional requirements specification is to be developed, then the requirements engineer manually copies and pastes the requirements documented in each use case's Specific Requirements along with the requirements in the Supplementary Requirements Specification. The approach is illustrated through an academic example of a microwave oven software product line.

A short paper by Probasco and Leffingwell [49] (S22) in a similar vein to that of Daniels et al. shows a Rational Software proposal to combine use cases and traditional requirements specifications through a simple construct called the *SRS Package*. This package pulls together the complete set of software requirements for a system, which may be contained in a single document, multiple documents, a requirements repository (consisting of the requirements' text, attributes and traceability), use case specifications, and the use case diagrams. Probasco and Leffingwell's document [49] is only a white paper and no concrete validation is mentioned.

4.2.8. Deriving requirements from user interface modelling

Finally, but of no less importance, a set of papers dealing with the generation of requirements documentation starting from user interface models exists. Jungmayr and Stumpe [33] (S23) propose extended *usage models* that consist of three sub-models: *scenario model*, *action model*, and *user interface model*: (1) the scenario model describes the semantics of the usage model in terms of goals that can be achieved when using the software, tasks that have to be accomplished in order to achieve the goals, and solutions, which are scenarios on how to use the software to solve a particular task; (2) the action model is a state machine that defines all possible sequences of user inputs (actions) and covers the information of a conventional usage model; and (3) the user interface model describes the user interface of the software and the interface elements that are themselves subject to user inputs. In this context, an HTML document can be automatically generated which structures the information already introduced in the extended usage models and can serve as user documentation. Extended usage models need a considerable effort to be built, but this is also true of conventional usage models. The structure of the output documents is shown in detail in [33]. Data in the output documents are linked by means of predefined string patterns. This work has been evaluated in a communication application to query commercial databases retrieving bibliographic references. The example is simple but well evaluated.

An old study by Smith [55] (S24) at MITRE Corporation presents a method to generate functional requirements concerning the user interface by starting from interface checklists. In these checklists the analyst indicates whether a concrete feature of the interface is *required*, *useful*, or *not needed*. The so-called *patterned prose* is used in order to generate the requirements in natural language by paraphrasing the questionnaires. In essence, patterned prose consists of a hierarchically related set of sentences, phrases and words, and their logical connectors, which are organized (and numbered) in correspondence with the structure of the checklists which are used to extract them. In our view, Smith's approach is very close to the design of the user interface: it seems closer to a *real* level of specification (physical) than to an *essential* (logical) level. Today applications have a GUI which is far more complex than those illustrated in the examples of this work, and we therefore believe that an extensive domain analysis would be needed to redefine the checklists. However, we found the idea of patterned prose interesting. An implementation in UNIX was available, but no validation procedures were reported in this paper.

4.2.9. Limitations of this SLR

As was previously mentioned, the searches in this SLR were defined by using certain overloaded software engineering terms, such as *model*, *requirement*, and *specification* as a starting point. It was necessary to use these general-use terms to seek studies in a heterogeneous field such as that defined in Section 2.1. The terms used to build the query (Section 2.2) have more synonyms, and some of these terms are homonyms. We cannot therefore guarantee that all the work related to the scope of this SLR has been found by means of the queries. For example, one important paper in this SLR (that of Meziane et al. [47] (S13)) was directly included in the review as a result of our previous knowledge of the papers of the field, and did not appear in the results of the searches. However, after checking the references to the papers included in the SLR, some of which have been published in top journals and conferences, we believe that we have analyzed the contents of an illustrative sample of the field.

As explained in Section 3.1, we have introduced two deviations from protocol in order to improve the completeness of the set of selected papers. We therefore believe that the discussion is more comprehensive, although the repeatability of the SLR is challenged.

We consider *Tool* and *Validation* to be part of the quality assessment criteria (Section 2.5), but these items are difficult to quantify with precision. Most of the automated support involved in the selected studies is not accessible and it is therefore difficult to precisely assess the tools' maturity level. What is more, the correctness of the validation procedures cannot be precisely evaluated from the reading of the papers.

5. Product requirements derivation in software product lines

While analyzing the selected papers we concluded that one weakness of this SLR was that no paper specifically addressed software product lines (SPLs). We knew of one commercial SPL tool, GEARS [3], which automatically derives a software requirements specification that describes in natural language the decisions made in the instantiation of the SPL variation points and the mandatory features of SPL products. We intuitively believed that SPLs constitute a domain in which the derivation of requirements from models can play an interesting role: product derivation in SPL implies navigation through a *decision space* (for example, a feature model [35] or an independent variability model [48]), solving a set of variation points to produce the specification of a product in the product line. We believed that the specification of this product in natural language in a requirements document could help clients to understand the configuration of the product they purchase. These reflections led us to extend the previous general SLR by performing a further SLR on product requirements derivation in SPLs. This new SLR is reported in full in this section.

Detailed processes for requirements modelling and guidance for requirements derivation are not sufficiently studied by the research community and are not handled by SPL tools [24]. In the SPL paradigm there is a consensus on the current need to pay greater attention to the product derivation process (see, for example, Käkölä and Dueñas' preface in [34]). In this vein Bühne et al. [13] present an overview of the research on the derivation of application specifications in SPLs. From a general point of view, they examine the development of an application requirements specification, paying particular attention to the treatment of stakeholders' new requirements which are not included in the SPL (called *deltas*), but without addressing the specific subject of this paper. This Section 5, in contrast, is devoted to the approaches that explicitly consider requirements derivation from the SPL variability models in application requirements engineering. Thus, a new research question arises:

RQ3. Which approaches take into account requirements derivation from SPL models?

The search strings shown in Table 5 were used to answer this question. The searches were performed in the Google Scholar search engine since it had delivered the most complete results in the general SLR. The inclusion and exclusion criteria were the same as those of the general SLR, but were obviously constrained to the domain of SPLs. The data collection and data analysis procedures were also the same. In this way 5 papers were selected (see Table 6). By following S25 in Table 6 we have considered interesting to include S26 in the selection (a deviation from protocol), and thus 6 studies were finally selected which are discussed in the remainder of this section. The combination mode of the 6 studies is generative and their scope is requirement. There is no study that addresses requirements documents generation. The initial models are feature and variability models (4 and 2 studies, respectively), while the target models are natural language (4 of the approaches) and formal notations (3 studies). Note that S27 addresses both formal notation and natural language.

Regarding product requirements derivation in SPLs, an early work by Hein et al. [30] (S25 in Table 6) claims that domain knowledge must be formalized to enable a partly automated requirements derivation process starting from a feature model. Hein et al. focus on a *feature model* (which extends that of FODA [35]), and a so-called *requirements model* (which encompasses the requirements texts and parameter definitions of the product or the domain). A requirement addresses variability through parameters, and each requirement that contains parameters can be seen as a *requirement template*. Features are modelled as the nodes of a tree. Basically, each node corresponds to a parameter. The feature tree therefore describes the possible values that can be assigned to a parameter. Traditional RMDB tools were designed to support single products, not product families, and these tools are thus out of the range of applicability to SPLs. Hein et al. report on several problems based on their experience in managing domain analysis work products with RMBD tools, especially with QSS DOORS. Furthermore, Hein et al. impose 9 requirements on a RMDB tool for SPLs, including "automatic generation of specific requirements texts". Generation of requirements specifications can be partially automated by the use of templates instantiated with different variants. This is regarded as very useful in the product line context, where different products and product variants have to be derived from abstract domain descriptions. A textual representation is more suitable for discussion with domain experts than more formal models in which the information is dispersed. These authors state that the application derivation problem can be handled by knowledge based systems, and especially configuration systems, such as that of Konwerk [53] (S26). These results are consolidated and validated in a real-scale industrial experiment at Bosch, in the Car Periphery Supervision (CPS) domain.

The previous work by Hein et al. imposes that the representation of domain knowledge must be formalized to enable a partly automated derivation process starting from the feature model. The authors also discovered that RMDB tools do not provide any customer guidance for product configuration in the SPL derivation process. We believe that a short paper by Rabiser et al. [50] (S27) can also be placed in this group. These authors present a tool suite, called DOPLER (*Decision-Oriented Product Line Engineering for effective Reuse*), which supports variability modelling and product configuration in SPLs. DOPLER includes a *ConfigurationWizard* to make decisions during product derivation, allowing product customization, requirements capture, and configuration generation. In a similar vein to that of Hein et al., Rabiser et al. claim that to fully exploit the benefits of SPLs it is essential to make product line models accessible to non-technicians such as sales people or even customers who make important decisions during product derivation. Decisions under the responsibility of a customer are listed as questions written in natural language, which are answered by yes/no. A graph and a tree-based view depict dependencies between decisions to support navigation in the decision space provided by the variability model. Newly captured requirements which are not yet covered by the SPL are also considered, and are described using the VOLERE template. Thus the product configuration generated from the variability model consists of a formalized tabular representation of the selected features together with the templates of the new textual requirements. The configurationWizard can also be used to launch simulator applications based on the selected assets. The references in this paper allow us to determine that this development research is taking place to support a typical scenario in product derivation and sales processes, regarding an SPL whose goal is the automation of continuous casting in SIEMENS VAI's steel plants.

Tavakoli and Reiser [57] (S28) propose a *requirements library* based on a new variability model to manage variability and commonality of product families at the requirements level. This study stems from a process improvement initiative at DaimlerChrysler in the automotive domain, where variability is complex due to the involvement of *hierarchical product lines*: model ranges of vehicles contain electronic control units which are, in turn, product lines. Requirements play a vital role in this industry, since the other software artefacts such as software architecture or code are developed by suppliers. Efficiency when creating specifications in short development cycles is of increasing importance. In this approach, deriving product specifications from the requirements library means a

Table 5
Search of SPL-related approaches (research question RQ3) in Google Scholar.

Search strings	Studies found	Candidate studies	Selected studies
+"requirements derivation" +"software product lines"	14	3	3
+"deriving requirements" +"software product lines"	6	2	2
+"requirements integration" +"software product lines"	4	0	0
+"integrating requirements" +"software product lines"	14	0	0

Table 6
Systematic review studies regarding RQ3.

ID	Author/s [ref.] (source)	Date	C. Mode		Scope		Initial model	Gen. Statem.		Method	Tool	Validation
			Gen.	Int.	Req.	Doc.		FL	NL			
Section 5. *Product's requirements derivation in software product lines*												
S25	Hein et al. [30] (gs)	2000	■	□	■	□	Feature model	□	■	Not specific	QSS DOORS	(ICS) Car Periphery Supervision at Bosch
S26	Schlick and Hein [53] (cited by S25)	2000	■	□	■	□	Feature model	□	■	Not specific	Konwerk configuration system	(ICS) Car Periphery Supervision at Bosch
S27	Rabiser et al. [50] (gs)	2007	■	□	■	□	Variability model	■	■ (V)	Not specific	DOPLER	(ICS) CL2 SPL for continuous casting in steel plants at SIEMENS VAI
S28	Tavakoli and Reiser [57] (gs)	2007	■	□	■	□	Variability model and requirements library	□	■	Not specific	Based on DOORS through a dxl plug-in	(ICS) Electronic control units at DaimlerChrysler
S29	Djebbi and Salinesi [22] (gs)	2007	■	□	■	□	Feature model (applicable to other SPL modelling lang.)	■	□	RED-PL	Ms Excel and Ms Excel Solver	(ICS) Blood analyzers at Stago Instruments
S30	Djebbi et al. [23] (gs)	2007	■	□	■	□	Feature model (applicable to other SPL modelling lang.)	■	□	RED-PL	On-going GNU-Prolog Solver-based prototype	(ICS) Blood analyzers at Stago Instruments

Source: gs (Google Scholar); *C. Mode* (Combination Mode): Gen.: Generative/Int.: Integrative; *Scope*: Req.: Requirement/Doc.: Document *Gen. Statem.* (Generated Statements): FL: Formal Language/NL: Natural Language; (V) Volere; *Validation*: (ACS) Academic Case Study/(ICS) Industrial Case Study/(IP) Industrial Practice; A-R: Action-Research.

stepwise decrease in variability. Requirements reuse within the requirements library is driven by selecting features, which in turn pre-select requirements in the database. Therefore the requirements engineers' main task shifts from developing requirements specifications to deriving requirements specifications (including the derivation of a variability definition). Tool support has been developed through the commercial RMDB tool DOORS [2] by using its script language *dxl* (*DOORS eXtension Language*). The authors state that the applicability of the approach in a tool is an important step toward the practical implementation of the approach. Although well-grounded in the automotive domain, it would appear that this work has not yet been applied in practice.

Djebbi and Salinesi's ongoing research into RED-PL (*Requirements Elicitation & Derivation for Product Lines*) [22] (S29) is aimed at defining a method for product requirements derivation in SPLs. Unlike previous approaches, RED-PL does not consist of selecting a product configuration from a SPL variability model in order to retrieve the requirements specifying the product to be built. In contrast, the RED-PL approach consists of (1) eliciting customers' requirements (customers are free to choose the way in which they specify requirements); (2) matching customers' requirements with SPL requirements, generating a set of wanted/unwanted requirements; and (3) deriving a consistent requirements collection that is optimal for a set of customers and company constraints (e.g. revenue, cost, resources, time). A constraint solver is proposed to match customers' needs, which are expressed textually, with the SPL requirements using similarity analysis techniques. SPL requirements are currently specified as feature diagrams although this approach is applicable to different SPL modelling languages (e.g. use cases, goals, UML, aspects). In [22], Djebbi and Salinesi use Integer Linear Programming (ILP) for similarity analysis. Ms Excel was used to apply ILP to a case study on blood analyzers developed in the STAGO Instruments Company. Some difficulties were observed while applying the method: (1) the experiment showed that ILP could not be used properly where complex requirements had to be expressed. Matching was difficult due to a lack of precision in the formulation of customer requirements. Difficulties were found not only with regard to terminology, but also conceptual mismatches between customers' requirements and SPL requirements (different levels of abstraction, different views). (2) ILP also has scalability problems. Furthermore, Djebbi et al. [23] (S30) is essentially a summary of [22] in which another constraint solver technique (Constraint Programming) is used, which shows better results in the case study. The authors report the ongoing development of a tool prototype via GNU-Prolog Solver. Although the authors do not explicitly define the term "requirement", it seems that a requirement in this context can be assimilated to a statement on the configuration of the SPL products. This work does not specifically address the generation of textual requirements. However, this approach has been analyzed because it introduces a different *modus operandi* and its output, an instantiated feature model, which we consider a kind of formal notation, can be easily presented in textual form.

6. Conclusions and further work

The literature on the generation of textual requirements and requirements documents starting from (business or software) models has been comprehensively reviewed and synthesized in this paper. It could be argued that this is a narrow topic within the mainstream research in requirements engineering. However, although the generation of requirements and documents from models has received relatively little attention in research (30 papers selected in the SLR), sound studies in literature have been found to corroborate a justification of interest in this line.

Moreover, as 17 of the 30 selected papers have been published in the last five years, we believe that it is reasonable to say that the interest in this topic, although narrow, is moderately increasing, especially in the SPL domain.

In our view, literate modelling – which we believe could broadly encompass all the work selected in this SLR – is an excellent idea which seems to have had little relevance in practice. What is more, the essence of this idea can be also found in a closely-related area: knowledge management. For instance, Eriksson [25] claims that "unfortunately, there is a surprisingly large gap between knowledge modelled in ontologies and the text documenting the same knowledge" and he proposes an approach to combine documents and ontologies, "allowing users to access the knowledge in multiple ways". We believe that there is value in making a stronger effort in the generation of textual requirements and documentation starting from models, in particular in literate modelling and SPLs, so that: (1) these approaches can be empirically evaluated in more industrial settings; (2) the set of techniques from which requirements and documents can be automatically generated are broadened (particularly covering the entire set of UML techniques related to requirements specification); (3) requirements engineering process models are refined to take into account the use of literate modelling; and (4) commercial RMDB tool support is developed. We believe that without proper tool support these approaches are not truly applicable in practice, especially in the context of agile developments.

We believe that both researchers and practitioners can benefit from an improvement in the readability of software engineering models, making these models available to a wider spectrum of stakeholders and thus improving their usability and facilitating their validation. Furthermore, regarding practitioners, one of the challenges that the technology roadmap of the industrial consortium ITEA has identified in relation to requirements-driven process management is precisely that special views are required that can give each stakeholder an appropriate vision of the requirements and how they are realised [32]. Moreover, the quality of the documentation generated by means of the literate modelling approach can serve practitioners in improving the vision of requirements documents as a contract between customers and developers. In addition, we believe that CASE developers can add value to their tools by including a literate modelling approach. When examining the evolution of software engineering in the last two decades, Sommerville [56] concludes that when looking beyond technology to the fundamental processes, much has stayed the same in software engineering. One of the reasons mentioned is precisely that CASE tools are still essentially diagram editors with some checking and code-generation functionality.

The results of the SLR have led us to propose five key issues that should be supported by a general-purpose RMDB tool which seamlessly integrates graphical and textual models of the system. In terms of the taxonomy defined in Section 3.2, the tool should be generative for both requirements and documents. In addition:

1. The tool should enable the automatic or closely monitored generation of requirements documents integrating (business and software) models and natural language requirements, ideally ensuring appropriate bidirectional traceability links between models and textual requirements.
2. The requirements documentation structure should follow the structure of the models in order to improve the requirements engineer's understanding: the section-subsection structure of the requirements documentation should emulate the models' structure in some way. Redundancy can be used to facilitate the understanding and/or modification of the documentation, but must be tracked by the tool.
3. Requirements documentation should be produced in a format that enables its modification. Requirements documents are living documents which should reflect open issues that may appear, change, and disappear. Therefore, formats that do not enable change, such as PDF, should not be exclusively used to browse textual requirements documents. These formats can obviously be used to create snapshots of the requirements specification under development, but we believe that the tool should manage the textual requirements in a manner which promotes change.
4. Once generated, the requirements documentation should be maintained in *synchronization* with the models, so that if an element changes in any of the two *views* of the system (the *models' view* or the *text view*), the tool will propagate the necessary changes to the related elements of the other view. Moreover, the requirements generation should be linked to an iterative and incremental development in such a way that the requirements generation will not necessarily affect the whole model, nor will further generations overwrite the changes directly effected in that documentation. A version control mechanism is therefore needed. If a change in the textual view does not correspond to any element in the models a *delta* must be registered and controlled.
5. In addition to key issue 2, the tool should enable the tailoring of the documentation according to its target readership or its intended use. The textual view, in particular, is not necessarily unique: there could be a *client contract view*, an *analyst view*, a *developer view*, etc.

There are a number of commercial tools that have links with the field reviewed in this paper. For instance, well-known commercial RMDB tools such as Requisite Pro [5], DOORS [2], and Caliber-RM [1] are integrated with the related UML-based tools of the suite to allow the synchronization of use case models and diagrams. Further work could therefore be carried out to review the tool market in order to analyze the combination between models and requirements in commercial RMDB tools.

Acknowledgements

Partially financed by the CICYT (*Science and Technology Joint Committee*), Spanish Ministry of Science and Technology (DEDALO Project, TIN2006-15175-C05-03), and the Science and Technology regional ministry of the Junta de Comunidades de Castilla-La Mancha (MELISA Project, PAC08-0142-335).

References

[1] Caliber-RM, Borland, 2009, <http://www.borland.com/caliber/index.html>.
[2] Doors, Telelogic (IBM Company), 2009, <http://www.telelogic.com/doors>.
[3] GEARS, BigLever Software, 2009, <www.biglever.com/overview.html>.
[4] OMG's MetaObject Facility (MOF), Object Management Group, 2009, <http://www.omg.org/mof/>.
[5] RequisitePro, IBM Rational Software, 2009, <http://www-306.ibm.com/software/rational/>.
[6] D. Alrajeh, A. Russo, S. Uchitel, Inferring operational requirements from scenarios and goal models using inductive learning, in: Intl. Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, ACM, Shanghai, China, 2006.
[7] A.I. Antón, C. Potts, The use of goals to surface requirements for evolving systems, in: 20th Intl. Conf. on Software Eng. (ICSE'98), IEEE Computer Society, Kyoto, Japan, 1998.
[8] J. Arlow, W. Emmerich, J. Quinn, Literate modelling – capturing business knowledge with the UML, in: «UML»'98: Beyond the Notation, First Intl. Workshop, Springer LNCS 1618, Mulhouse, France, 1998.
[9] J. Arlow, I. Neustadt, Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML, The Addison-Wesley Object Technology Series, Addison-Wesley, Boston, 2004.
[10] B. Berenbach, The automated extraction of requirements from UML models, in: 11th Intl. Conf. on Requirements Eng. (RE'03), IEEE Computer Society, Monterey, CA, USA, 2003.

[11] B. Berenbach, Comparison of UML and text based requirements engineering, in: Companion to the 19th Conf. on OO Programming, Sys., Lang., and App. (OOPSLA'04), ACM Press, Vancouver, BC, Canada, 2004.

[12] J. Biolchini, P. Gomes Mian, A.C. Cruz Natali, G. Horta Travassos, Systematic Review in Software Engineering, TR-ES 679/05, Sys. Eng. and Computer Sci. Rio de Janeiro, Dep. COPPE/UFRJ, 2005.

[13] S. Bühne, G. Halmans, K. Lauenroth, K. Pohl, Scenario-based application requirements engineering, in: Software Product Lines: Research Issues in Engineering and Management, Springer, Berlin, 2006, pp. 161–194.

[14] D. Burke, K. Johannisson, Translating formal software specifications to natural language: a grammar-based approach, in: Logical Aspects of Computational Linguistics Conf., Bordeaux, France, 2005.

[15] G. Cabral, A. Sampaio, Formal specification generation from requirement documents, Electron. Notes Theor. Comput. Sci. 195 (2008) 171–188.

[16] CMMI, CMMI, Capability Maturity Model Integration, v.1.2, 2006, <http://www.sei.cmu.edu/cmmi/general/index.html>.

[17] K. Cox, K.T. Phalp, S.J. Bleistein, J.M. Verner, Deriving requirements from process models via the problem frames approach, Inform Software Technol. 47 (5) (2005) 319–337.

[18] B.H.C. Cheng, J.M. Atlee, Research directions in requirements engineering, in: Future of Software Eng. (FOSE'07), Minneapolis, USA, 2007.

[19] J. Daniels, R. Botta, T. Bahill, A hybrid requirements capture process, in: INCOSE 15th Annual Intl. Symposium on Sys. Eng., Rochester, NY, 2005.

[20] A.M. Davis, in: Just Enough Requirements Management: Where Software Development Meets Marketing, Dorset House, New York, NY, 2005.

[21] R. De Landtsheer, E. Letier, A. van Lamsweerde, Deriving tabular event-based specifications from goal-oriented requirements models, Requirements Eng. 9 (2) (2004) 104–120.

[22] O. Djebbi, C. Salinesi. RED-PL, a method for deriving product requirements from a product line requirements model, in: 19th Intl. Conf. on Advanced Inf. Sys. Eng. (CAiSE'07), Trondheim, Norway, 2007.

[23] O. Djebbi, C. Salinesi, D. Diaz, Deriving product line requirements: the RED-PL guidance approach, in: Asia–Pacific Software Eng. Conf. (APSEC 2007), Nagoya, Japan, 2007.

[24] O. Djebbi, C. Salinesi, G. Fanmuy, Industry survey of product lines management tools: requirements, qualities and open issues, in: 15th IEEE Intl. Requirements Eng. Conf. (RE '07), Delhi, India, 2007.

[25] H. Eriksson, The semantic-document approach to combining documents and ontologies, Int. J. Human–Comput. Stud. 65 (7) (2007) 624–639.

[26] A. Finkelstein, W. Emmerich, The future of requirements management tools, in: Inf. Sys. in Public Administration and Law, 2000.

[27] D. Firesmith, Modern requirements specifications, J. Object Tech. 2 (1) (2003) 53–64.

[28] D. Firesmith, Generating complete, unambigous, and verifiable requirements from stories, scenarios, and use cases, J. Object Tech. 3 (10) (2004) 27–39.

[29] R.F. Goldsmith, Discovering Real Business Requirements for Software Project Success, Artech House Publishers, Boston, London, 2004.

[30] A. Hein, J. MacGregor, M. Schlick, Requirements and feature management for software product lines. in: Deutscher Software-Produktlinien Workshop (DSPL-1), Kaiserslautern, Germany, 2000.

[31] IEEE, Std 830-1998, Guide to Software Requirements Specifications, Resource and Technique Standards, vol. 4, The Institute of Electrical and Electronics Engineers, Inc., IEEE Software Eng. Stds. Collection, 1999.

[32] ITEA-Office, ITEA Technology Roadmap for Software Intensive Systems, second ed., May 2004, <http://www.itea-office.org>.

[33] S. Jungmayr, J. Stumpe, Another motivation for usage models: generation of user documentation, in: CONQUEST'98, Nüremberg, Germany, 1998.

[34] T. Käkölä, J.C. Dueñas (Eds.), Software Product Lines. Research Issues in Engineering and Management, Springer, Berlin Heidelberg, 2006.

[35] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, SEI (Software Eng. Inst.), Carnegie Mellon Univ., Pittsburgh, PA, 1990.

[36] B.A. Kitchenham, Guidelines for performing Systematic Literature Reviews in Software Engineering, EBSE Tech. Report, EBSE-2007-01, Software Eng. Group, School of Computer Sci. and Math., Keele Univ. (UK), Dep. of Computer Sci., Univ. of Durham (UK), 2007.

[37] B.A. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering – a systematic literature review, Inform Software Technol. 51 (1) (2009) 7–15.

[38] C. Larman, Applying UML and Patterns, third ed., Prentice Hall, Upper Saddle River, NJ, 2005.

[39] B. Lavoie, O. Rambow, E. Reiter, The ModelExplainer, in: 8th Intl. Workshop on Natural Lang. Generation (INLG'96), Herstmonceux Castle, England, 1996.

[40] E. Letier, A. van Lamsweerde, Deriving operational software specifications from system goals, in: 10th Symposium on Foundations of Software Eng. 2002 (FSE'02), ACM Press, Charleston, South Carolina, USA, 2002.

[41] N. Maiden, S. Jones, C. Ncube, J. Lockerbie, Using i* in requirements projects: some experiences and lessons, in: E. Yu (Ed.), Social Modeling for Requirements Engineering, MIT Press, 2007.

[42] N. Maiden, S. Minocha, K. Manning, M. Ryan, CREWS-SAVRE: systematic scenario generation and use, in: 3rd Intl. Conf. on Requirements Eng. (ICRE'98), IEEE Computer Society, Colorado Springs, CO, USA, 1998.

[43] N. Maiden, C. Ncube, S. Kamali, N. Seyff, P. Grünbacher, Exploring scenario forms and ways of use to discover requirements on airports that minimize environmental impact, in: 15th Intl. Req. Eng. Conf. (RE'07), New Delhi, India, 2007.

[44] N. Maiden, S. Robertson, Developing use cases and scenarios in the requirements process, in: 27th Intl. Conf. on Software Eng. (ICSE '05), ACM Press, St. Louis, MO, USA, 2005.

[45] N.A.M. Maiden, S. Manning, S. Jones, J. Greenwood, Generating requirements from systems models using patterns: a case study, Requirements Eng. 10 (4) (2005) 276–288.

[46] A. Mavin, N. Maiden, Determining socio-technical systems requirements: experiences with generating and walking through scenarios, in: 11th Intl. Conf. on Requirements Eng. (RE'03), IEEE Computer Society, Monterey, CA, USA, 2003.

[47] F. Meziane, N. Athanasakis, S. Ananiadou, Generating natural language specifications from UML class diagrams, Requirements Eng. 13 (1) (2008) 1–18.

[48] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering, Foundations, Principles and Techniques, Springer, Berlin Heidelberg, 2005.

[49] L. Probasco, D. Leffingwell, Combining software requirements specifications with use case modeling, in: Rational White Paper, 1999.

[50] R. Rabiser, D. Dhungana, P. Grunbacher, K. Lehner, C. Federspiel, involving non-technicians in product derivation and requirements engineering: a tool suite for product line engineering, in: 15th IEEE Intl. Requirements Eng. Conf. (RE '07), Delhi, India, 2007.

[51] B. Regnell, E. Kamsties, V. Gervasi, Summary of the 10th anniversary workshop on requirements engineering: foundation for software quality, in: Requirements Eng.: Foundation for Software Quality 2004 (REFSQ'04), Riga, Latvia, 2004.

[52] S. Robertson, J. Robertson, Mastering the Requirements Process, second ed., Addison-Wesley, New York, NY, 2006.

[53] M. Schlick, A. Hein, Knowledge engineering in software product lines, in: European Conf. on Artificial Intelligence (ECAI 2000), Workshop on Knowledge-Based Sys. for Model-Based Eng., Berlin, Germany, 2000.

[54] N. Seyff, F. Graf, P. Grünbacher, N. Maiden, The mobile scenario presenter: a tool for in situ requirements discovery with scenarios, in: 15th Intl. Requirements Eng. Conf. (RE'07), New Delhi, India, 2007.

[55] S.L. Smith, Patterned prose for automatic specification generation, in: Conf. on Human Factors in Computing Sys., ACM Press, Gaithersburg, Maryland, USA, 1982.

[56] I. Sommerville, Software Engineering, seventh ed., Pearson Education Limited, Boston, 2004.

[57] R. Tavakoli, M.O. Reiser, Reusing requirements: the need for extended variability models, in: Intl. Symposium on Fundamentals of Software Eng. (FSEN 2007), Tehran, Iran, 2007.

[58] O. Türetken, O. Su, O. Demirörs, Automating software requirements generation from business process models, in: 1st Conf. on the Principles of Software Eng. (PRISE'04), Buenos Aires, Argentina, 2004.

[59] A. van Lamsweerde, Goal-oriented requirements engineering: a roundtrip from research to practice, in: 12th Requirements Eng. Conf. 2004 (RE'04), IEEE Publishers, Kyoto, Japan.

[60] A. van Lamsweerde, L. Willemet, Inferring declarative requirements specifications from operational scenarios, IEEE Trans. Software Eng. 24 (12) (1998) 1089–1114.

[61] E. Yu, P. Du Bois, E. Dubois, J. Mylopoulos, From organization models to system requirements. A cooperating agents approach, in: 3rd Intl. Conf. on Cooperative Inf. Sys. (CoopIS-95), Vienna, Austria, 1995.

[62] K. Zachos, N. Maiden, A. Tosar, Rich Media Scenarios for Discovering Requirements, IEEE Software 22 (5) (2005) 89–97.