Information and Software Technology 56 (2014) 821–838

Contents lists available at ScienceDirect

Information and Software Technology

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

Formal verification of static software models in MDE: A systematic review

Carlos A. González *, Jordi Cabot

AtlanMod Research Group, École des Mines de Nantes – INRIA, LINA, 4, Rue Alfred Kastler, F-44307 Nantes Cedex 3, France

ARTICLE INFO

Article history:
Received 5 July 2013
Received in revised form 17 February 2014
Accepted 12 March 2014
Available online 19 March 2014

Keywords:
MDE
Formal verification
OCL
Systematic literature review

ABSTRACT

Context: Model-driven Engineering (MDE) promotes the utilization of models as primary artifacts in all software engineering activities. Therefore, mechanisms to ensure model correctness become crucial, specially when applying MDE to the development of software, where software is the result of a chain of (semi)automatic model transformations that refine initial abstract models to lower level ones from which the final code is eventually generated. Clearly, in this context, an error in the model/s is propagated to the code endangering the soundness of the resulting software. Formal verification of software models is a promising approach that advocates the employment of formal methods to achieve model correctness, and it has received a considerable amount of attention in the last few years.

Objective: The objective of this paper is to analyze the state of the art in the field of formal verification of models, restricting the analysis to those approaches applied over static software models complemented or not with constraints expressed in textual languages, typically the Object Constraint Language (OCL).

Method: We have conducted a Systematic Literature Review (SLR) of the published works in this field, describing their main characteristics.

Results: The study is based on a set of 48 resources that have been grouped in 18 different approaches according to their affinity. For each of them we have analyzed, among other issues, the formalism used, the support given to OCL, the correctness properties addressed or the feedback yielded by the verification process.

Conclusions: One of the most important conclusions obtained is that current model verification approaches are strongly influenced by the support given to OCL. Another important finding is that in general, current verification tools present important flaws like the lack of integration into the model designer tool chain or the lack of efficiency when verifying large, real-life models.

Contents

* Corresponding author. Tel.: +33 (0)2 51 85 82 16.
E-mail addresses: carlos.gonzalez@mines-nantes.fr (C.A. González), jordi.cabot@mines-nantes.fr (J. Cabot).

http://dx.doi.org/10.1016/j.infsof.2014.03.003
0950-5849/

1. Introduction

Model-driven Engineering (MDE) is an approach that promotes the utilization of models as primary artifacts in all software engineering activities. In a MDE-based software development process, the software is not coded by hand, but by designing and creating a number of models to be successively and (semi)automatically transformed into more refined models, and eventually into code comprising the new software system. When MDE approaches are used to develop complex systems, the complexity of models involved increases, thus turning creating and editing them into error-prone tasks. This complexity endangers the MDE development process and the soundness of the resulting software.

Ensuring software correctness is not a new challenge, though. On the contrary, it is an old challenge that software engineers continue to struggle with [1]. Thanks to the efforts made by the research community, different trends have appeared to try to address the problem. One of them is software verification, which comprises those approaches based on the utilization of formal methods and formal analysis techniques to prove software correctness. The word "formal" states that these methods are based on mathematical theories. One of the major advantages of using formal methods to verify software is to avoid the introduction of imprecision or ambiguity in the process. In these approaches, it is typical to proceed by going through two different stages. In the first one, the formalization stage, the problem to be solved is represented using one of these mathematical theories or formalisms. In the second one, the reasoning stage, the resolution of the formalized problem is addressed by utilizing tools specialized in reasoning over the chosen formalism.

Considering the problems that model complexity has brought into MDE-based software development processes and the advantages of the utilization of formal methods when verifying software, it comes as no surprise that a lot of effort has been made in studying how to apply formal methods and formal analysis techniques to ensure model correctness. This has obviously led to a significant amount of research results.

While working on [2], we had the opportunity of making a first analysis of some of this research. It was hard for us to grasp a clear view, each work using their own terminology to describe the problem, in some cases, borrowed from what we considered were different fields, like consistency checking or software validation. Because of this, we considered that, in order to check whether this initial impression of ours was right, a deeper analysis was needed. In this regard, although some systematic literature reviews had been conducted to analyze notions like model quality [3] or the consistency between models [4], we did not know of the existence of studies devoted to the analysis of the utilization of formal methods and formal analysis techniques to ensure model correctness.

It is also true, though, that addressing this type of exhaustive analysis in its full generality, and without considering factors like model diversity, is probably inadequate, leading for example to the comparison of approaches that use models of very different nature, and impractical, since it would probably produce cumbersome results. Because of these factors, the systematic literature review presented in this paper focuses on providing an exhaustive analysis and comparison of the research initiatives done in the field of formal verification of static models, only. The reason for this is that static models are arguably the models more commonly adopted at the time of describing the specification of a software system.

Static models, also commonly referred to as structural models, are those models used to represent, totally or partially, the structure of a system design, that is, those models that show a time independent view of the system. Regarding this, the most popular static model is the UML[1] (Unified Modeling Language) class diagram, although in this study, other models that fall outside of the UML umbrella, like for example entity-relationship models or models developed by using Domain Specific Modeling Languages (DSMLs), were also covered. Also related to the type of models covered in this study, it is important to notice that since UML class diagrams exhibit certain limitations to represent precisely detailed aspects of a system, they are often used in combination with OCL[2] (Object Constraint Language). OCL provides additional capabilities to enrich models, like for example, a mechanism for the definition of integrity constraints that the instances of a model must hold. The analysis of how OCL is supported by the existing model verification approaches was also covered as part of this study. It is also worth remarking that model correctness refers to the ability of the model under analysis to satisfy one or more correctness properties (like for example satisfiability). Obviously, another important part of this study is to analyze what correctness properties are supported by every verification approach.

More specifically, this study addressed the following research questions:

- RQ1: What are the typical phases in model verification approaches?

[1] http://www.omg.org/spec/UML/.
[2] http://www.omg.org/spec/OCL/.

- RQ2: What are the formal methods and techniques employed in the verification approaches?
- RQ3: What are the types of static models that can be verified in each case?
- RQ4: Up to what extent is the Object Constraint Language (OCL) supported?
- RQ5: Which correctness properties are verified in each case?
- RQ6: Are the existing approaches automatic (in the sense that user intervention is not required to steer the verification process) and complete?
- RQ7: Are the existing approaches supported by a verification tool?
- RQ8: What type of feedback is obtained from the verification process in each case?

To answer them, we designed a systematic procedure to retrieve all the works related to formal verification of static models published since 2002. The retrieval process led to the discovery of 48 works which were analyzed individually and then grouped into 18 different studies. These groups arose naturally, since the majority of works were part of coarse grained studies made up by more than one work, and was done attending to factors like the works goals, their authors and the relationship of the work with other related works. Once these 18 studies were compiled, their contents were carefully analyzed looking for the information relevant for the study's goals.

This paper is organized as follows: Section 2 describes how the systematic review has been conducted, including the data sources consulted, the inclusion/exclusion criteria, quality assessment and data extraction criteria. Section 3 shows the results obtained. In Section 4 we talk about those aspects that could be considered as limitations or threats for the study's validity. Section 5 is devoted to discuss the findings of the study. Finally, we draw some conclusions in Section 6.

2. Exploration method

As stated in [5,6], a systematic review, also known as systematic literature review, is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. A systematic review is driven by a review protocol, which specifies, among other things, the research questions to be addressed, the mechanisms to search and identify the relevant research works and how the data collected are synthesized to carry out the review's goals.

In the rest of this section we describe the different steps we took to carry out our review.

2.1. Inclusion and exclusion criteria

Ensuring that the research works collected were aligned with the objectives of the systematic review required the definition of some criteria to determine both, which works were considered relevant, and which ones fell outside of the scope of this review.

Regarding the criteria to determine relevant works, i.e. the inclusion criteria, the main one was that only works addressing the formal verification of static properties over static models, enriched or not with OCL constraints, could be considered. With this in mind, works must either (i) discuss/present verification tools, or (ii) address formal verification from a theoretical point of view, in order to be part of the study.

On the other hand, we also defined some exclusion criteria to identify irrelevant works. In particular, papers that (i) addressed verification issues by means of techniques like model checking [7], typical of scenarios involving dynamic or behavioral models, (ii) involved the verification of dynamic or behavioral properties over static models, (iii) were devoted to consistency checking or validation issues, or (iv) focused on business models, fell outside the scope of this study and therefore had to be rejected. Moreover, another limitation applied was that only works written in English and published since 2002 were taken into account.

Table 1
Repositories used for the analysis.

Repository (URL)	Acronym
DBLP (http://www.dblp.org/search/index.php)	DBLP
Faceted DBLP (http://dblp.l3s.de)	FDBLP
Web of Knowledge (http://www.webofknowledge.com)	WOK
Science Direct (http://www.sciencedirect.com)	SD
SpringerLink (http://www.springerlink.com)	SPL
IEEE Xplore (http://ieeexplore.ieee.org)	IEEE
CiteSeerX (http://citeseerx.ist.psu.edu/index)	CITE

2.2. Data sources

To begin with the process of collecting the research works relevant to this study, it was necessary to determine what digital libraries, and electronic databases to use as sources. It is typical that the selection of digital libraries or electronic databases is driven by factors such as their popularity, frequency of actualization, facilities to run complex queries or the number of works indexed. In the case of this systematic review, although these factors were indeed considered, there was an additional one that had to be taken into account, which was the fact that the repository had to be equipped with mechanisms to facilitate the batch retrieval of bibliographical references. The reason for this was simple. As it will be detailed in the following subsections, the process of determining the set of relevant works was composed by a number of pruning stages, each one consisting in discarding a certain amount of irrelevant works until, finally, obtaining the relevant ones out of the last pruning stage. In order to conduct the pruning as easiest as possible, we intended to load the works from the digital libraries into a reference manager[3] and proceed with it from there, so we needed digital libraries that either allowed downloading hundreds of citations quickly and easily, or permitted the utilization of bots, spiders or, at least, download managers to automate the retrieval of citations as much as possible.

Having identified what we needed from the digital libraries, we analyzed a fairly comprehensive list of them, finally choosing the ones listed in Table 1. Although it did not fulfill the requisite of facilitating citations retrieval mentioned earlier, SpringerLink digital library was necessary to get access to the citations of certain issues of Springer journals, that were not present in the rest of digital libraries considered. The repositories discarded can be seen in Table 2, along with the reasons (Table 3) why they were dismissed.

2.3. Search strategy: initial set of works

Once the digital libraries to be used were determined, we could start the process leading to the obtainment of the set of relevant works. The first stage of this process was to obtain what we called the initial set of works, that is, the works to be loaded into the reference manager and over which the pruning stages were going to be conducted.

To compile the initial set of works, we designed a strategy consisting in primary running ten searches in the scope of a selected list of journals, international conferences and international

[3] The reference manager we used in this study was EndNote X5. For further information on this, the reader can visit http://www.endnote.com.

Table 2
Repositories discarded for the analysis and the reasons.

Repository (URL)	Reason
Google Scholar (http://scholar.google.com/)	1, 2
Scopus (http://www.scopus.com)	4
ACM Digital Library (http://dl.acm.org/)	1
Microsoft Academic Search (http://academic.research.microsoft.com/)	1, 2
Cornell University Library (http://arxiv.org/)	3

Table 3
Reasons to discard a repository.

Number	Reason
1	Absence of facilities to batch download references in a suitable format
2	Limitations to build structured searches
3	Limited contents compared to other repositories
4	Not having access

workshops, and then, in order to complement the results obtained with other relevant works that might have been published out of the journals, conferences and workshops in that list, running the same ten searches over the full contents of one of the digital libraries considered. The reasons why we designed ten searches (Table 10) were twofold. On the one hand, we were able to gather within ten searches the search terms we considered most relevant for the study. On the other hand, it was necessary to establish a limit on, first, the time spent conducting these searches and, second, on the number of records retrieved in order to make the forthcoming stages more manageable. Once we designed the searches, it was the moment to determine what journals (Table 4), conferences (Table 5) and workshops (Table 6) were relevant to our purposes. We did this relying on our own expertise and knowledge of the domain.

When we put this strategy in motion, we noticed that not all the digital libraries considered facilitated building complex searches limited to the scope of a given conference or workshop. When trying to build these types of queries in certain digital libraries (SpringerLink, IEEE Xplore or Science Direct), it was necessary to know the name of the publication series where the proceedings of the conference/workshop had been published, thus complicating looking for works just by using the conference/workshop name. In other cases, even though the digital libraries (DBLP, Faceted DBLP) allowed us discriminating works by conference/workshop in an easy way, the search language was not expressive enough to build the queries we needed to conduct our searches. Among the digital libraries considered, only "Web Of Knowledge" (WOK) allowed us to conduct complex searches limited to the scope of a given conference/workshop in an easy way. However, WOK did not index all the conferences and workshops in Tables 5 and 6, so at this point we realized that due to the differences in the querying systems of the digital libraries, we simply could not run our searches homogeneously and load the results into the reference manager. Hence, in order to proceed keeping the searches over the journals, conferences and workshops as homogeneous as possible, we decided to retrieve all the bibliographical references (not only those ones closely related to the study's goals) published since 2002 in the conferences, workshops and journals selected, load them into the reference manager grouped by their origin (journal, conference or workshop) and then conduct the ten searches from there, using the reference manager search facilities.

Tables 7–9 display the number of works retrieved and loaded into the reference manager for each of the journals, conferences and workshops considered. They also show the digital libraries and search strings employed in every case. Due to not all the sources considered have been around since 2002, the information in the tables is complemented with the time interval enclosing the works retrieved. In those cases where the search strings do not include any clause restricting the search by date, this restriction was applied by further refining the search results, using facilities present in the GUI of the digital library used. By doing this, we loaded 31,853 bibliographical references into our reference manager, 20,214 coming from the list of journals, 11,060 corresponding to works published in the list of selected conferences, and 579 obtained from the list of workshops. This 31,853 bibliographical references constituted the search space where we conducted the ten searches that yielded the first subset of the initial set of works.

With the reference manager loaded, we used its search facilities to conduct the searches shown in Table 10 over each of the three groups of citations. The results obtained, as we mentioned at the beginning of the section, were complemented by running the same ten searches over one of the selected digital libraries (we chose the IEEE Xplore digital library for this due to its flexible querying system), thus forming the initial set of records of the study. Table 11 shows the results obtained. The initial set of works was composed by 8079 citations. It is important to mention that the use of a reference manager greatly facilitated the discovery of overlaps in the searches conducted over the journals, conferences and workshops. There was no easy way, though, to do the same with the results

Table 4
Journals considered in the study.

Conference	Acronym	Publisher	Time span (Volume, Issue)
Transactions on Software Engineering and Methodology	TOSEM	ACM	January'02(V11, I1)–July'12(V21, I3)
Software And System Modeling	SoSyM	Springer	September'02(V1, I1)–October'12(V11, I4)
Acta Informatica	Acta	Springer	January'02(V38, I4)–November'12(V49, I8)
Constraints	Constraints	Springer	January'02(V7, I1)–October'12(V17, I4)
Science of Computer Programming	SCP	Elsevier	January'02(V42, I1)–February'13(V78, I2)
Computer	Computer	IEEE	January'02(V35, I1)–November'12(V45, I11)
Electronic Notes in Theoretical Computer Science	ENTCS	Elsevier	January'02(V61)–December'12(V290)
Information and Software Technology	IST	Elsevier	January'02(V44, I1)–February'13(V55, I2)
International Journal of Software Engineering and Knowledge Engineering	IJSEKE	World Scientific	February'02(V12, I1)–September'12(V22, I6)
Transactions on Software Engineering	TSE	IEEE	January'02(V28, I1)–December'12(V38, I6)
Software	Software	IEEE	January'02(V19, I1)–December'12(V29, I6)
IBM Systems Journal	Systems	IBM	January'02(V41, I1)–October'08(V47, I4)
IBM Journal of Research and Development	IBM RD	IBM	January'02(V46, I1)–December'12(V56, I6)
Data And Knowledge Engineering	DKE	Elsevier	January'02(V40, I1)–December'12 (V82)
Journal of Object Technology	JOT	–	May'02(V1, I1)–August'12(V11, I2)
Journal of Systems and Software	JSS	Elsevier	January'02(V60, I1)–January'13(V86, I1)
Formal Aspects of Computing	FOC	Springer	May'02(V13, I2)–July'12(V24, I6)

Table 5
International conferences considered in the study.

Conference	Acronym	Time span (editions)
International Conference on Model Driven Engineering Languages and Systems	MODELS	2005(8th)–2012(15th)
International Conference on Automated Software Engineering	ASE	2002(17th)–2012(27th)
The Unified Modeling Language, International Conference	UML	2002(5th)–2004(7th)
Symposium On Applied Computing	SAC	2002(17th)–2012(27th)
European Conference on Modeling Foundations and Applications – European Conference on Model Driven Architecture – Foundations and Applications	ECMFA ECMDA-FA	2005(1st)–2012(8th)
International Conference on Conceptual Modeling	ER	2002(21st)–2012(31st)
Tools	TOOLS	2008(46th)–2012(50th)
International Conference on Quality Software	QSIC	2003(3rd)–2012(12th)
International Conference on Software Engineering	ICSE	2002(24th)–2012(34th)
European Software Engineering Conference/Symposium on the Foundations of Software Engineering	ESEC/FSE	2002(FSE 10th)–2011(ESEC 2011/FSE 19th)
International Conference on Software Testing, Verification and Validation	ICST	2008(1st)–2012(5th)
International Conference on Integrated Formal Methods	iFM	2002(3rd)–2012(9th)
International Conference on Advanced Information Systems Engineering	CAiSE	2002(12th)–2012(24th)
International Symposium on Software Reliability Engineering	ISSRE	2002(13th)–2011(22nd)
International Symposium on Formal Methods	FM	2002(11th)–2012(18th)
International Conference on Fundamental Approaches to Software Engineering	FASE	2002(5th)–2012(15th)
International Conference on Model Transformation	ICMT	2008(1st)–2012(5th)

Table 6
International workshops considered in the study.

Workshop	Acronym	Time span (editions)
International Workshop on Description Logics	DL	2002(15th)–2012(25th)
OCL Workshop	OCL	2003(3rd)–2011(11th)[a]

[a] 2004 and 2005 Editions have been disregarded because their proceedings are not available in the digital libraries considered.

Table 7
Journals retrieval. Repositories searched.

Journal	Digital library	Records found	Interval	Search query
TOSEM	WOK	161	January'02–July'12	SO = (ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY)
SoSyM	WOK	189	March'07–October'12	SO = (SOFTWARE AND SYSTEMS MODELING)
	SPL	123	September'02–December'06	Manually
Acta	WOK	270	January'02–November'12	SO = (ACTA INFORMATICA)
Const	WOK	190	January'03–October'12	SO = (CONSTRAINTS)
	SPL	19	January'02–July'02	Manually
SCP	SD	940	January'02–February'13	pub-date > 2001 and src (SCIENCE OF COMPUTER PROGRAMMING)
Comp	IEEE	3928	January'02–November'12	(''Publication Title'':''computer'')
ENTCS	SD	4421	January'02–December'12	pub-date > 2001 and src (Electronic Notes in Theoretical Computer Science)
IST	SD	1237	January'02–February'13	pub-date > 2001 and src (Information and Software Technology)
IJSEKE	WOK	514	February'02–September'12	SO = (INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING)
TSE	IEEE	1234	January'02–December'12	(''Publication Title'':''Software Engineering, IEEE Transactions on'')
Software	IEEE	1756	January'02–December'12	(''Publication Title'':''software, ieee'')
Systems	IEEE	402	January'02–October'08	(''Publication Title'':''ibm systems'')
IBM RD	IEEE	739	January'02–October'12	(''Publication Title'':''IBM Journal of Research and Development '')
DKE	SD	994	January'02–December'12	pub-date > 2001 and src (Data & Knowledge Engineering)
JOT	FDBLP	766	May'02–August'12	Journal of Object Technology (Venues only)
JSS	SD	1987	January'02–January'13	pub-date > 2001 and src (Journal of Systems and Software)
FOC	WOK	277	2004–2012	SO = (FORMAL ASPECTS OF COMPUTING)
	SPL	67	May'02–December'03	Manually

obtained out of the searches conducted over the IEEE Xplore digital library.

2.4. Paper selection

After we obtained the initial set of 8079 citations and had them loaded into the reference manager, we could start with the pruning stages, that is ruling out those works not related with the study's goals. The pruning tasks we applied to the initial set of citations can be seen in the following list:

- Duplicates deletion.
- Delete references with no author.
- Keyword pruning.
- Title pruning.
- Abstract pruning.

The first stage was deleting duplicated citations. Taking into account the way we proceeded to obtain the initial set of works, duplicated citations could come from two different sources: the overlaps in the results coming from the ten searches conducted over the IEEE Xplore digital library, and the overlaps between these

Table 8
Conferences retrieval. Repositories searched.

Conference	Digital library	Records found	Search query	Results refinement
MODELS	FDBLP	455	MoDELS (Venues Only)	2005–2012
ASE	FDBLP	876	ASE (Venues Only)	2002–2012
UML	FDBLP	104	UML (Venues Only)	2002–2004
SAC	FDBLP	3792	SAC (Venues Only)	2002–2012
ECMDA-FA	FDBLP	128	ECMDA_FA (Venues Only)	2005–2009
ECMFA	FDBLP	95	ECMFA (Venues Only Exact Match)	2010–2012
ER	FDBLP	522	ER (Venues Only)	2002–2012
TOOLS	FDBLP	109	TOOLS (Venues Only Exact Match)	2002–2012
QSIC	FDBLP	541	QSIC (Venues Only)	2003–2012
ICSE	FDBLP	1669	ICSE (Venues Only Exact Match)	2002–2012
ESEC/FSE	FDBLP	583	sigsoft fse (Venues Only)	2002–2012
ICST	FDBLP	352	ICST (Venues Only Exact Match)	2008–2012
iFM	FDBLP	182	iFM (Venues Only)	2002–2012
ISSRE	FDBLP	381	ISSRE (Venues Only)	2002–2011
CAiSE	FDBLP	511	CAISE (Venues Only Exact Match)	2002–2012
FM	FDBLP	246	FM (Venues Only)	2005–2012
FME	FDBLP	85	FME (Venues Only)	2002–2003
FASE	FDBLP	336	FASE (Venues Only)	2002–2012
ICMT	FDBLP	93	ICMT (Venues Only)	2008–2012

Table 9
Workshops retrieval. Repositories searched.

Workshop	Digital library	Records found	Interval	Search query
DL	DBLP	507	2002–2012	Description Logics (Venues Only Exact Match)
OCL	DBLP	11	2003	Venue: Electr. Notes Theor. Comput. Sci. (ENTCS) author: Peter H. Schmitt year: 2004
	DBLP	11	2006	Venue: eceasst author: Dan Chiorean year: 2006
	DBLP	11	2007	Venue: eceasst author: Steffen Zschaler year: 2008 preface
	DBLP	10	2008	Venue: eceasst author: Jordi Cabot year: 2008
	DBLP	8	2009	Venue: eceasst author: Tony Clark year: 2009
	DBLP	10	2010	Venue: eceasst author: Mira Balaban year: 2010
	DBLP	11	2011	Venue: eceasst author: Robert Clarisó year: 2011

Table 10
Search strings.

Search string stem	
(“VERIFYING” OR “VERIFICATION” OR “CHECKING” OR “CORRECTNESS” OR “CONSISTENCY” OR “REASONING” OR “SATISFIABILITY”) AND ...	
N.	Search string suffixes
S01	UML
S02	MODEL AND (“DOMAIN” OR “CONCEPTUAL” OR “STATIC” OR “EMF”)
S03	DIAGRAM
S04	ENTITY-RELATIONSHIP
S05	ENTITY RELATIONSHIP
S06	DSL
S07	LANGUAGE AND (“DOMAIN” OR “SPECIFIC”)
S08	“ER”
S09	OCL
S10	SCHEMA

Table 11
Initial set of records for the systematic review.

Search	Journals	Conferences	Workshops	IEEE Xplore
S01	148	46	4	873
S02	315	8	2	4126
S03	134	15	2	1851
S04	3	0	0	66
S05	0	0	1	154
S06	8	0	0	106
S07	417	0	0	1733
S08	4	2	0	127
S09	37	10	3	90
S10	51	3	1	433
Overlap	293	22	3	0
Total	824	62	10	7183

results and those coming from the ten searches conducted over the selected journals, conferences and workshops. EndNote X5, the reference manager we used, has a powerful mechanism for the automatic detection of duplicates in which it is even possible to instruct the tool to determine when a given citation must be considered a duplicate of one another. In this case, we set up the tool to consider duplicates those citations in which the fields, “title”, “author” and “publication year” were identical. Once the tool was configured, we simply ran the duplicates search process obtaining 1579 duplicate records that were eliminated.

With the initial set of works reduced to 6500 records the next step was deleting those references with no author. At this point of the process, there was a little group of citations that actually did not refer to any concrete work but to journal front covers, tables of contents, conference proceedings and so forth. A commonality among these citations was the absence of author, so in order to get rid of them, we simply searched citations with no author and ruled out the results yielded, thus reducing the set of records in another 51 records. At this point of the process the set of records under analysis was composed by 6449 citations.

The next stage was possibly the most sensitive one. Since we were looking for papers devoted to a very specific type of model verification (formal verification of static software models), instead of choosing a set of keywords related to the study’s goals and collect those citations in which any of these keywords were present, we proceeded the other way around. We identified a set of keywords that, in our opinion, should not be representative of any work devoted to the formal verification of static models, then

looked for the works where these keywords were present, and ruled out the records obtained. In order to choose the right set of keywords, we initially compiled a set of approximately 100 keywords and conducted searches to see the amount of papers in which each of them were present. Then, based on the results obtained, we reduced the set of keywords to 50 by excluding both, those ones too generic that appeared in thousands of works, and those ones too specific that appeared in just a handful of them. Table 12 shows the 50 keywords, along with the number of works in which each of these keywords were present and how many of them were ruled out. The difference between the records yielded by each search and the records that were ruled out obeys to the overlap in the searches, that is, the fact that there were many records in which at least two of the keywords of the list were present at the same time. After conducting the 50 searches corresponding to the selected keywords, we finally ruled out 5386 records, thus reducing the set of works to 1063 works.

After performing the keyword pruning stage it was the time for the title pruning stage. Before describing this, it is important to state that, from this point forward, the forthcoming pruning stages were conducted manually. In the particular case of the title pruning stage, although it might have been possible to conduct it in an automated manner, by searching the titles for certain words or fixed-expressions, we feared that this strategy, in case of a high variability on these titles, might have ruled out some relevant works. Besides, the previous pruning stages had reduced the number of relevant works to a number slightly over 1000 (1063 to be exact), and we considered that reviewing the titles of, roughly speaking, 1000 papers was affordable and worth the effort.

Regarding the process itself, it was conducted as follows: the first author read the titles and elaborated two lists: one with the papers that should be discarded and one with the ones to be kept. The second author read then these two lists and expressed his opinion. In those cases where authors disagreed on what the right list was for a given title, that work was automatically placed on the list of works to be kept. We acted in a conservative manner here because more information apart from the title would be available on the next pruning stage, thus allowing us to make a more informed decision. Finally, we ruled out 737 records, therefore keeping a set of 326 works to be analyzed in the last step.

To finish with the pruning stages, we read the abstracts of the works resulting from the title pruning stage. In the end, we discarded 304 papers. The reason why we ruled out so many works in this stage is that we were too conservative in the title reviewing stage, so we ended up with many papers that after a quick glance to their abstracts resulted not to be related to formal verification of static models at all. In addition to this, an important set of papers devoted to model checking or consistency checking were also discarded. In the first case, the papers about model checking were related to the verification of model dynamic aspects. In the second, the papers were about checking consistency between different models, usually involving models dealing with dynamic aspects, like state machines or state charts.

Finally, after all these pruning stages, we got the set of 22 relevant works that can be seen in Table 13. This set of papers was complemented with those obtained out of the actions described in the next subsection, thus giving way to the final set of relevant papers considered for analysis in this systematic review.

2.5. Quality assessment

The last step we carried out to complete the list of relevant works was the quality assessment stage. The quality assessment consisted in first, carefully reading the 22 papers obtained up to this point and, second, performing a process of snowballing, paying attention to the "Related Work" section and bibliographical references of the collected works. With this we wanted to ensure that the works collected fulfilled the inclusion criteria described in Section 2.1, and also to complement the list of collected works, looking for those ones that although relevant for the objectives of the review, had not been discovered yet.

It is important to mention that one of the risks when snowballing is the exponential growth in the number of papers to be read, since the process must be recursively repeated with any new paper added to the list. We took two measures to control this. The first

Table 12
Records pruned by keyword.

Keyword	Records		Keyword	Records	
	Found	Pruned		Found	Pruned
DATA	1220	1220	FREQUENCY	235	25
ENGINE	898	722	BUSINESS	222	9
TIME	870	638	LEARNING	215	22
CONTROL	728	388	CHIP	213	15
NETWORK	617	280	METRIC	207	28
PROCESSING	616	190	IMAGE	190	32
SIMULATION	584	215	AGENT	185	25
INFORMATION	558	139	PROTOCOL	183	20
CIRCUIT	558	194	SAFETY	178	29
KNOWLEDGE	539	160	ELECTRIC	177	16
WEB	522	131	MOBILE	176	12
ARCHITECTURE	498	120	WAVE	174	11
DYNAMIC	482	95	MATH	172	18
TESTING	443	127	SEARCH	170	15
SERVICE	424	22	COMPILER	144	29
REQUIREMENT	377	79	AUTOMATA	134	11
POWER	316	41	GUI	132	7
PATTERN	304	45	OPTIMIZATION	127	12
SECURITY	293	23	METHODOLOGY	117	9
SIGNAL	289	33	VIRTUAL	114	12
FLOW	286	21	MEDICAL	112	6
COMMUNICATION	269	20	CONFIGURATION	111	7
TEMPORAL	269	44	DIAGNOSIS	111	9
GRAPHIC	238	26	PARALLEL	104	6
ELECTRONIC	237	20	EQUATION	98	8

Table 13
First group of relevant works obtained.

Ref.	Title
[8]	Checking Full Satisfiability of Conceptual Models
[9]	Full Satisfiability of UML Class Diagrams
[10]	Complexity of Reasoning in Entity Relationship Models
[11]	Reasoning Over Extended ER Models
[12]	A UML-Based Method for Deciding Finite Satisfiability in Description Logics
[13]	Reasoning on UML Class Diagrams is EXPTIME-hard
[14]	UMLtoCSP: a Tool for the Formal Verification of UML/OCL Models Using Constraint Programming
[15]	Verification of UML/OCL Class Diagrams using Constraint Programming
[16]	Lightweight String Reasoning for OCL
[17]	Finite Satisfiability of UML Class Diagrams by Constraint Programming
[18]	Checking Unsatisfiability for OCL Constraints
[19]	USE: A UML-Based Specification Environment For Validating UML and OCL
[2]	EMFtoCSP: A tool for the lightweight verification of EMF models
[20]	Efficient Reasoning About Finite Satisfiability of UML Class Diagrams with Constrained Generalization Sets
[21]	OCL-Lite: Finite reasoning on UML/OCL conceptual schemas
[22]	AuRUS: Automated Reasoning on UML/OCL Schemas
[23]	Reasoning on UML Class Diagrams with OCL Constraints
[24]	Decidable Reasoning in UML Schemas with Constraints
[25]	Verification and Validation of UML Conceptual Schemas with OCL Constraints
[26]	Mapping from OCL/UML metamodel to PVS metamodel
[27]	Verification-driven slicing of UML/OCL models
[28]	Formal Semantics and Reasoning about UML Class Diagram

one was to respect the exclusion criteria described in Section 2.1. That is, we did not include in the process the references to those papers that had been published before 2002. The second one was to rule out those references that, although relevant for the paper under evaluation, clearly fell outside of the scope of our analysis. This way, relying on our own experience, we ruled out an important number of references corresponding to tools or aimed at describing auxiliary techniques.

After reading the papers, we confirmed that all of them indeed satisfied the inclusion criteria and, additionally, the process of snowballing yielded the list of 26 works in Table 14 that, after being carefully read for confirmation, were also included in the set of relevant works (in Section 5 we discuss possible reasons why more works were collected out of the snowballing process than out of the structured searches).

Finally, after the pruning and quality assessment stages we ended up with a list of 48 relevant works that constituted the object of analysis of this study. Fig. 1 summarizes the evolution of the resulting set of works obtained, as we went through the stages leading to its obtainment.

With the quality assessment stage finishes the part of the study corresponding to determining the list of relevant works, but before continuing, and for the sake of the interested reader, we would like to say a word about the time employed on the stages described thus far. Determining and retrieving the works that constituted the search space for this study took approximately three months. This included the analysis of repositories, selecting the conferences, journals and workshops of interest, analyzing several reference managers, and, most importantly, testing and trying different strategies to automate the retrieval process as much as possible.

Table 14
Additional relevant works obtained from the process of snowballing.

Ref.	Title
[29]	An Accessible Formal Specification of the UML and OCL Meta-Model in Isabelle/HOL
[30]	On Challenges of Model Transformation From UML to Alloy
[31]	UML2Alloy: A Challenging Model Transformation
[32]	Translating the Object Constraint Language into First-order Predicate Logic
[33]	Using DLs to Reason on UML Class Diagrams
[34]	Reasoning on UML Class Diagrams
[35]	An MDA Framework Supporting OCL
[36]	The HOL-OCL Book
[37]	HOL-OCL: A Formal Proof Environment for UML/OCL
[38]	Towards Implementing Finite Model Reasoning in Description Logics
[39]	Finite Model Reasoning on UML Class Diagrams via Constraint Programming
[40]	A Formal Framework for Reasoning on UML Class Diagrams
[41]	Satisfiability of Object-Oriented Database Constraints With Set and Bag Attributes
[42]	Finite Satisfiability of Integrity Constraints in Object-Oriented Database Schemas
[43]	Validating UML and OCL models in USE by Automatic Snapshot Generation
[44]	Proving and Visualizing OCL Invariant Independence by Automatically Generated Test Cases
[45]	Consistency, Independence and Consequences in UML and OCL Models
[46]	Using B Formal Specifications for Analysis and Verification of UML/OCL Models
[47]	From UML and OCL to Relational Logic and Back
[48]	Strengthening SAT-Based Validation of UML/OCL Models by Representing Collections as Relations
[49]	Extensive Validation of OCL Models by Integrating SAT Solving into USE
[50]	OCL-Lite: A Decidable (Yet Expressive) Fragment of OCL
[51]	Mapping UML Models Incorporating OCL Constraints into Object-Z
[52]	Providing Explanations for Database Schema Validation
[53]	Verifying UML/OCL Models Using Boolean Satisfiability
[54]	Encoding OCL Data Types for SAT-Based Verification of UML/OCL Models

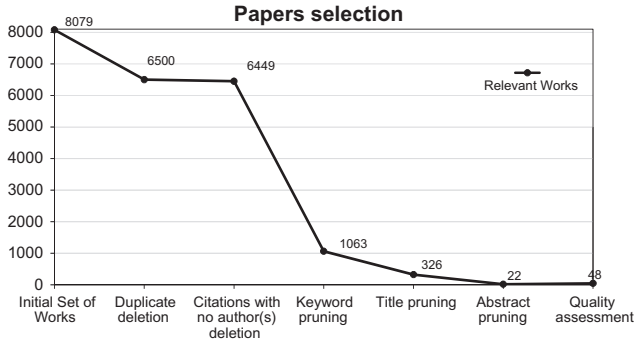

Fig. 1. Evolution of the set of relevant works.

Choosing the initial set of works took approximately three weeks. The majority of this time was spent designing and testing multiple search queries. With the search queries already in place, obtaining the initial set of works took us approximately one day. Deleting duplicates and references without author took no time. That was immediate thanks to the reference manager facilities. The keyword pruning stage took approximately one week. Almost all of this time was spent looking for the most suitable catalog of keywords. After having obtained that catalog, the pruning took a couple of hours. The title pruning stage took approximately two days, and the abstract pruning stage some more time, about one week. Finally, the snowballing process took approximately one month.

2.6. Data extraction

After having identified the list of relevant works, the next step consisted in designing a strategy to extract from them the information needed to answer the research questions defined in Section 1. To accomplish this, we elaborated, through a series of consensus meetings, an Excel spreadsheet where we gathered together all the necessary information. In particular, we structured the information in the following way: we saved one row in the spreadsheet for every relevant work, and then we added as many columns as the number of different pieces of information gathered. In particular, we collected two different groups of data items, the first group corresponding to those data items needed to identify a given work, including the creation of a unique identifier (Table 15), and then a second group corresponding to the data items needed to answer the research questions (Table 16).

It is important to remark that not all the papers considered provided enough information to fill each of the columns in the spreadsheet. This is because some of the papers are devoted to specific aspects of the verification process, such as model formalization or tool presentation. Overall, all the information needed could be figured out when those works were identified as to be part of broader studies described throughout several papers.

3. Results

After describing the process followed to collect the works considered relevant for this study, it is the time of briefly describing them and presenting the results of the analysis conducted to answer the research questions listed in Section 1.

3.1. Presentation of the selected papers

As it was mentioned in the introduction, we grouped the 48 relevant works (Tables 13 and 14) in 18 different studies taking into account factors like the existence of clear relationships between some papers or their authors. In this subsection, we present these

Table 15
Data items extracted for identification purposes.

Data item	Description
ID	Unique identifier
Title	Title of the paper
Author(s)	Author(s) of the paper
Year	Year of publication

Table 16
Data items extracted to answer the research questions.

Data item	Description
Goal	Brief description of the paper goal
Model	Type of model supported
OCL	Degree of support of OCL constraints in the verification process
Formalization	Formalism employed (if any) for the verification process
Properties verified	Properties verified
Termination	Whether the verification process ensures termination or not
Completeness	Completeness of the verification process
Automation	Was the verification process automatic or interactive?
Tool	Name of the verification tool implementing the approach (if any)
Solver	Name of the underlying solver employed (if any)
Feedback	Results yielded by the verification process

Table 17
Studies identified and the corresponding citations.

Study	Representative name	Refs.
S1	UMLtoCSP	[14,15,27]
S2	EMFtoCSP	[2,16]
S3	FiniteSat	[20,12]
S4	AuRUS	[22–25,52]
S5	DL	[39,17,38,8,9,11,10,13,34,33,40]
S6	OCL-Lite	[50,21]
S7	OODB	[42,41]
S8	HOL-OCL	[37,35,36]
S9	UML2Alloy	[31,30]
S10	USE	[43,19,45,44,49,48,47]
S11	BV-BSAT	[53,54]
S12	PVS	[26]
S13	KeY	[32]
S14	Object-Z	[51]
S15	UML-B	[46]
S16	CDOCL-HOL	[29]
S17	MathForm	[28]
S18	OCL2FOL	[18]

18 studies, that for the sake of clarity have been summarized in Table 17.

It is important to remark that in some cases, the relevant works conforming the 18 studies presented here are based on previous works that either were published before 2002 or fell outside the scope of this study for some other reason. In those occasions in which we considered it necessary for facilitating the contextualization of a certain study, we included citations to those works in its corresponding description.

The first study (S1) comprises the works directly related to UMLtoCSP [14]. UMLtoCSP, developed by Cabot et al. [14,15], is a Java tool for the formal verification of UML/OCL models based on constraint programming. The tool works by translating a class diagram along with its OCL constraints and the desired verification properties into a Constraint Satisfaction Problem (CSP). In particular, the tool builds a Constraint Logic Program (CLP) which is then fed into a solver called ECL^{i}PS^e Constraint Programming System[4] for its resolution. Although UMLtoCSP was designed with the intent of supporting OCL constraints in its full generality, as of this study, not all the OCL constructs were supported. Even though, the tool features some notable characteristics: user intervention during the reasoning process is not required (i.e. it is automatic) and termination is ensured. This is possible because the tool follows a bounded verification approach (the user must define the size of the search scope beforehand). The drawback of bounded verification approaches, though, is that results are only conclusive if a solution to the CSP is found (i.e. it is not complete). Regarding correctness properties, UMLtoCSP supports the verification of strong satisfiability, weak satisfiability, liveliness of a class, lack of constraints subsumption and lack of constraint redundancies.

The tool was later on complemented with the work of Shaikh et al. [27] consisting in the development of a slicing technique for UML/OCL class diagrams. The presence of this technique turned UMLtoCSP into a more efficient tool when verifying weak satisfiability or strong satisfiability.

The second study (S2) gathers the works around EMFtoCSP [2], which is an Eclipse[5] integrated tool for the verification of EMF models annotated with OCL constraints. EMFtoCSP represents an evolution of the UMLtoCSP tool described in [S1], and can be used to verify a larger variety of models, noticeably, EMFtoCSP is well-suited for the verification of Domain Specific Languages (DSL). Compared to UMLtoCSP, EMFtoCSP includes, among other improvements, a revisited version of the CSP generation mechanism. In particular, and thanks to the work of Büttner et al. [16], EMFtoCSP supports the analysis of OCL expressions including operation on Strings in general terms.

The third study (S3) collects the works of Azzam Maraee and Mira Balaban, who developed a linear programming based method for reasoning about finite satisfiability of UML class diagrams with constrained generalization sets [20]. In the authors' words, finite satisfiability is the problem of deciding whether a given class has a finite, non-empty extension in some model. Their method builds on top of the work of Lenzerini and Nobili [55], which is based on the transformation of the cardinality constraints into a set of linear inequalities whose size is polynomial in the size of the diagram. This way, the finite satisfiability problem is reduced to the problem of finding a solution to a system of linear inequalities. The algorithm proposed, called "FiniteSat", was later on improved [12] to handle all the types of constraints included in an enhanced version of the Description Logics to class diagrams translation presented in [34].

The fourth study (S4) congregates the verification works based on the CQC Method [56], a mechanism to perform query containment tests on deductive database schemas, that has also been used to determine properties like satisfiability or predicate liveliness over this type of schema. In this regard, Queralt et al. presented AuRUS [22], a tool for assessing the semantic quality of a conceptual schema consisting in a UML class diagram complemented with OCL arbitrary constraints, which extends SVTe [57], a relational database schema validation tool. Apart from the satisfiability of the conceptual schema, AuRUS can verify liveliness of classes or associations and redundancy of constraints, without requiring user intervention during the reasoning process. The tool works [23] by first translating both, the class diagram and the OCL constraints into a set of first-order formulas that represent the structural schema, and then verifying, by using the CQC Method, whether the supported properties hold. In the case that the properties do not hold, the tool is able to give the user a hint about the changes of the schema that are needed to fix the problem identified [52]. AuRUS does not guarantee termination when dealing with general OCL

[4] http://eclipseclp.org.
[5] http://www.eclipse.org/.

expressions, but it does so [24] when dealing with a specific subset of constraints.

Finally, in [25] the authors presented several improvements, like an enhanced version of the conceptual schema to logic translation, or refinements on the mechanism presented in [24] which is used by AuRUS to determine whether the reasoning process will terminate or not.

The fifth study (S5) compiles the work developed by Calvanese et al. on reasoning over entity-relationship models and UML class diagrams since the year 2002. Related to this, Cadoli et al. [39,17,38] developed an approach to encode the problem of finite model reasoning (i.e. checking whether a class is forced to have either zero or infinitely many objects) in UML class diagrams as a CSP that is solved by relying on the use of off-the-shelf tools for constraint modeling and programming. These works exploit the encoding of class diagrams in terms of Description Logics proposed by Berardi et al. [13,34,33] and Cali et al. [40], to take advantage of the finite model reasoning techniques developed for Description Logics in [58], based on reducing the problem of reasoning on a Description Logics knowledge base to the problem of finding a solution for a set of linear inequalities.

Moreover, the work of Berardi et al. [13,34] has also served as a basis for the complexity analyses conducted by Artale et al. [8,9,11,10] which have established important results about the problem of verifying full satisfiability over different variants of UML class diagrams or entity-relationship models.

The sixth study (S6) contains the work related to OCL-Lite [50,21], a fragment of OCL that ensures termination and completeness when reasoning on UML conceptual schemas enriched with arbitrary constraints within the bounds of this fragment. Apart from the identification of such a fragment, the authors propose an encoding of UML class diagrams enriched with constraints within its bounds in Description Logics. In this regard, they take advantage of the works developed by Calvanese et al. that have been described in [S5]. Finally, they show how it is possible to use existing reasoners to provide reasoning support to check properties like schema satisfiability or constraint redundancy over these models.

The seventh study (S7) refers to the work of Anna Formica on checking finite satisfiability of database constraints. In particular, a decidable graph-theoretic approach to finite satisfiability checking is proposed in [42]. This approach, which is limited to integrity constraints involving comparison operators, was later on expanded in [41] to cover cardinality constraints among others. In both cases, the database schemas are described using fragments of TQL^{++} [59], an object-oriented data definition language aimed at modeling the structural aspects and integrity constraints of object-oriented database models [60].

The eighth study (S8) is about the formal proof environment HOL-OCL [37], developed by Brucker and Wolff. HOL-OCL is integrated into a framework supporting a formal model-driven engineering process, which is described in [35], and works by automatically encoding the class diagram along with the OCL constraints in Higher-Order Logics (HOL). This encoding, which is described in detail in [36], can then be used to reason over the model by means of the interactive theorem prover Isabelle [61]. A drawback of this approach, though, is that it generally requires user-interaction to steer the reasoning process.

The ninth study (S9) makes reference to the works about UML2Alloy [31,30], which is the name of a tool developed by Anastakasis et al., that can be used to check the satisfiability of a UML class diagram enriched or not with OCL constraints. UML2Alloy, as it can be inferred from its name, works by transforming the model to be verified into the relational logic of Alloy[6] [62], which is then fed into the SAT solvers embedded within the Alloy Analyzer. Regarding the verification process in itself, UML2Alloy, as it is the case of UMLtoCSP and EMFtoCSP, follows a bounded verification approach, that is, the user must establish the boundaries of the search space where looking for a solution (i.e. the approach is not complete).

The tenth study (S10) relates to the work of Gogolla et al. around the USE tool [43,19]. USE was originally conceived as a validation tool, but it has evolved significantly throughout the years and now it supports model verification as well. In one of its initial versions [43], the tool worked by generating instances, or fragments of instances of a given input model (these instances or fragments are called snapshots in the USE terminology), to be checked, one by one, against a series of OCL invariants. This version of the tool was able to support model verification to a certain extent, like for example, to check constraint independence as shown in [45,44]. However, since the original approach presented the problem of the enumerative nature of the snapshot generator, it was later on improved by Kuhlmann et al. [49,48,47] with the development of a mechanism to translate UML and OCL concepts into relational logic, which uses the SAT-based constraint solver KodKod [63] for the reasoning stage. This way, the original snapshot generator was replaced by a more efficient SAT-based bounded search and generation of snapshots fulfilling the user-specified constraints, thus expanding the capabilities of the tool to perform model verification tasks.

The eleventh study (S11) compiles the work of Soeken et al. [53,54] on the verification of UML/OCL models. In their proposed approach a verification problem is encoded into a bit-vector formulation and fed into a SMT/SAT solver for its verification. A verification problem is made up by three different elements: the system state, a series of OCL constraints, and the verification task. The system state is an encoding of the attribute assignments in every object as well as of the links between these objects, and the verification task is nothing but the encoding of the property to be verified. Since general OCL constraints are supported, the method follows a bounded verification approach to ensure termination, that is, limits in the number of objects and associations as well as in the domains of the attributes are enforced.

The twelfth study (S12) is limited to the approach proposed by Lukman Ab. Rahim [26] to transform UML class diagrams and OCL constraints into the specification language of the PVS[7] (Prototype Verification System) theorem prover [64]. PVS is based on HOL and comes with a specification language that allows writing theorems to prove a given specification. The approach focuses on describing how a set of rules written using the Epsilon Transformation Language[8] [65] maps UML class diagrams and OCL elements into a proposed PVS metamodel. The idea is to serialize the model into a PVS specification to be fed into the PVS theorem prover for its analysis. It is important to remark that although the mapping proposed does not exclude in advance any type of OCL expression, certain operations in the OCL standard library cannot be mapped due to PVS limitations.

The thirteenth study (S13) refers to the work of Beckert et al. [32] on the formalization of UML class diagrams with OCL constraints into dynamic logic, which is a multi-modal extension of first-order logic (FOL). The approach, that has been implemented in Java and focuses only on the formalization stage, is part of the KeY system [66], a software development tool[9] that seeks the integration of design, implementation, formal specification and formal verification of object-oriented software.

The fourteenth study (S14) relates to the work of Roe et al. [51] on the mapping of UML class diagrams and OCL constraints into a

[6] http://alloy.mit.edu/alloy/.

[7] http://pvs.csl.sri.com/.

[8] http://www.eclipse.org/epsilon/.

[9] http://www.key-project.org/.

formal specification described with Object-Z [67,68]. Object-Z is an extension of the Z specification language [69] to facilitate the construction of specifications in an object-oriented style. The mapping, which is described informally, is based on the work of Kim et al. [70], where a formal semantic mapping between the two languages is provided.

The fifteenth study (S15) is about the work of Marcano and Levy [46] describing a systematic translation of UML class diagrams and OCL constraints into a B formal specification. The B specification language [71] provides a means for developing mathematically proven software and systems, through the use of rigorous mathematical reasoning. The approach works by first deducing a B abstract specification from the UML class diagrams, which is then complemented with the addition of a number of B formal expressions generated out of the OCL constraints. The approach is supported by the implementation of a prototype tool where the analysis of the generated B specification relies on the utilization of the Atelier-B[10] tool. Although this approach is primarily intended for consistency checking purposes, it can also be used to check verification properties like the presence of contradictions in constraints (incoherent constraints).

The sixteenth study (S16) covers the work of Ali et al. [29] on the formalization of UML class diagrams and OCL constraints into HOL. The work shares some similarities with that of Brucker and Wolff described in [S8], as the intent of reasoning over the resulting encoding using the theorem prover Isabelle [61]. The main difference, though, is the utilization of simpler techniques to build the formalization, so that it can be more accessible to practitioners of the software industry.

The seventeenth study (S17) comprises the work of Marcin Szlenk [28] on the formalization of UML class diagrams into a mathematical notation based mainly on the utilization of sets and partial functions. The formalization is used to outline the subject of reasoning about a class diagram, introducing the formal definition of consistency of a classifier, which is similar in concept to the verification property "liveliness of a class" mentioned in [S1].

The eighteenth study (S18) is about the work of Clavel et al. [18] on formalizing and reasoning over OCL constraints. In this work, a mapping from a subset of OCL into FOL is proposed with the intent of supporting verification using automated reasoning tools like Prover9,[11] an automated theorem solver, and Yices,[12] a SMT solver. In particular, the authors propose reasoning on their own notion of (unbounded) unsatisfiability of OCL constraints over a class diagram.

3.2. Result analysis

After presenting the 18 studies grouping the relevant works gathered, we now move to answer the research questions posed in Section 1. Regarding this, the answers provided here are complemented by Tables 18–20, in which a summary of the most relevant characteristics of the 18 studies can be found.

3.2.1. RQ1: What are the typical phases in model verification approaches?

Throughout the analysis of the 18 studies we have clearly identified two different stages that can occur during model verification. The first stage is that of formalization. It is typical for the model to be verified, along with its OCL constraints (if supported) and the correctness properties, to be represented in some kind of formalism which is then exploited during the reasoning process to determine whether the model holds those correctness properties. This reasoning process is the second stage and it is usually conducted with the help of solvers or tools specialized in reasoning over the chosen formalism. In general, all the studies describe some sort of formalization [S5, S16, S17] being examples of this, but additionally, some of them also cover the reasoning stage either in an applied way, by means of presenting a verification tool [S1, S2, S4, S9, S11], or in a more theoretical way [S5, S7].

Table 18
Formalization techniques used in each study.

Study	Formalization technique
S1 (UMLtoCSP)	CSP
S2 (EMFtoCSP)	CSP
S3 (FiniteSat)	System of Linear Inequalities
S4 (AuRUS)	FOL
S5 (DL)	Description Logics, CSP
S6 (OCL-Lite)	Description Logics
S7 (OODB)	TQL^{++}
S8 (HOL-OCL)	HOL
S9 (UML2Alloy)	Relational Logic
S10 (USE)	Relational Logic
S11 (BV-BSAT)	Bit-vector Logic
S12 (PVS)	HOL
S13 (KeY)	Dynamic Logic
S14 (Object-Z)	Object-Z
S15 (UML-B)	B
S16 (CDOCL-HOL)	HOL
S17 (MathForm)	Mathematical Notation
S18 (OCL2FOL)	FOL

3.2.2. RQ2: What are the formal methods and techniques employed in the verification approaches?

As it was mentioned earlier, in all the studies analyzed, some sort of formalization stage takes place. In this regard, the most typical way of formalizing is by means of some kind of logical representation like FOL, DL, or HOL, among others. Some examples of studies following this formalization approach are [S6, S8, S9, S18, S11, S13]. Specification languages like B or Object-Z have also been used, as it is the case in [S14, S15]. Finally, it is also popular to encode the problem of model verification as a CSP [S1, S2, S5] or by means of other mathematical notations [S3, S17].

3.2.3. RQ3: What are the types of static models that can be verified in each case?

All the studies presented here address the verification of UML class diagrams, that arguably, are the diagrams most typically used when representing the structure of a software system. [S2] also supports Eclipse EMF models, thus supporting the verification of models developed by using Domain Specific Modeling Languages (DSMLs). Database schemas in the form of entity-relationship diagrams [S5] or object-oriented database schemas [S7] have also been object of analysis.

3.2.4. RQ4: Up to what extent is the Object Constraint Language (OCL) supported?

When it comes to support OCL, there are three types of approaches, namely: those ones that try to support OCL in its full generality, as it is the case in [S1, S2, S4, S10, S13, S14] among others, those ones that support a certain subset of OCL [S6, S12, S18], or those ones that do not support OCL at all [S3, S5, S7, S17]. Typically, the level of support is strongly correlated to factors like automation or the completeness of the approach, as it is shown in the answer to RQ6 later on.

3.2.5. RQ5: Which correctness properties are verified in each case?

All the correctness properties addressed in the different studies fall into two big groups: properties about the instantiability of the model and properties about the relationship among constraints.

[10] http://www.atelierb.eu/en/.
[11] http://www.cs.unm.edu/mccune/mace4/.
[12] http://yices.csl.sri.com/.

Table 19
Types of model and properties covered in each study.

Study	Model type	OCL constraints	Property
S1 (UMLtoCSP)	Class Diagrams	Yes, General	Strong Satisfiability, Weak Satisfiability, Liveness of a Class, Constraint Redundancy, Constraint Subsumption
S2 (EMFtoCSP)	Class Diagrams, EMF Models	Yes, General	Strong Satisfiability, Weak Satisfiability, Liveliness of a Class, Constraint Redundancy, Constraint Subsumption
S3 (FiniteSat)	Class Diagrams	No	Finite Satisfiability
S4 (AuRUS)	Class Diagrams	Yes, General	Satisfiability of the Conceptual Schema, Liveliness of Classes or Associations, Redundancy of Constraints
S5 (DL)	Class Diagrams	No	Finite Satisfiability
S6 (OCL-Lite)	Class Diagrams	Yes, Subset	Schema Satisfiability, Class Satisfiability, Constraint Redundancy
S7 (OODB)	Object-Oriented DB Schemas	No	Database Constraints Satisfiability
S8 (HOL-OCL)	Class Diagrams	Yes, General	Satisfiability
S9 (UML2Alloy)	Class Diagrams	Yes, General	Weak Satisfiability, Strong Satisfiability, Liveliness of a Class
S10 (USE)	Class Diagrams	Yes, General	Satisfiability, Constraint Independence
S11 (BV-BSAT)	Class Diagrams	Yes, General	Satisfiability, Constraint Independence
S12 (PVS)	Class Diagrams	Yes, Subset	N/A
S13 (KeY)	Class Diagrams	Yes, General	N/A
S14 (Object-Z)	Class Diagrams	Yes, General	N/A
S15 (UML-B)	Class Diagrams	Yes, General	Incoherent Constraints
S16 (CDOCl-HOL)	Class Diagrams	Yes, General	N/A
S17 (MathForm)	Class Diagrams	No	Consistency of a Classifier
S18 (OCL2FOL)	Class Diagrams	Yes, Subset	Unsatisfiability of OCL Constraints

Table 20
Tools, execution mode and feedback provided in each study.

Study	Tool's name	Feedback	Automation	Completeness	Termination ensured
S1 (UMLtoCSP)	UMLtoCSP	Yes/No + Sample model when "Yes"	Automatic	No	Yes
S2 (EMFtoCSP)	EMFtoCSP	Yes/No + Sample model when "Yes"	Automatic	No	Yes
S3 (FiniteSat)	N/A	N/A	N/A	N/A	N/A
S4 (AuRUS)	AuRUS	Yes/No + Sample model when "Yes" + Hint when "No"	Automatic	Decidability analysis dependent	Only for specific OCL constraints
S5 (DL)	N/A	N/A	N/A	N/A	N/A
S6 (OCL-Lite)	N/A	N/A	N/A	Yes	Yes
S7 (OODB)	N/A	N/A	N/A	Yes	Yes
S8 (HOL-OCL)	HOL-OCL	Yes/No	Interactive	Yes	No
S9 (UML2Alloy)	UML2Alloy	Yes/No + Sample model when "Yes"	Automatic	No	Yes
S10 (USE)	USE	Yes/No + Sample model when "Yes"	Automatic	No	Yes
S11 (BV-BSAT)	Prototype	Yes/No + Sample model when "Yes"	Automatic	No	Yes
S12 (PVS)	N/A	N/A	N/A	N/A	N/A
S13 (KeY)	N/A	N/A	N/A	N/A	N/A
S14 (Object-Z)	N/A	N/A	N/A	N/A	N/A
S15 (UML-B)	Prototype	Yes/No	Interactive	Yes	No
S16 (CDOCL-HOL)	N/A	N/A	N/A	N/A	N/A
S17 (MathForm)	N/A	N/A	N/A	N/A	N/A
S18 (OCL2FOL)	Prototype	Yes/No	Yes	Solver dependent	Solver dependent

Regarding the properties of the first group, the most important one is satisfiability, that is to check whether it is possible to create legal instances of the model. Satisfiability comes in different variants: strong satisfiability means that the legal model instance must include instances of all the classes and associations in the model, weak satisfiability, on the other hand is less strict, and it does not enforce the instantiation of all classes and associations. The verification of this property in its different variants is addressed in [S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11]. Other property that falls into this category is "liveliness of a class", that is to check whether it is possible to create legal instances of the model including at least one instance of a certain class. This verification property, which can be seen as a particular case of satisfiability, is addressed in [S1, S2, S4, S9]. When it comes to the second group of properties, it is typical to check the presence of redundant constraints [S1, S2, S4, S6], although there are other properties like "constraint subsumption" [S1, S2] or "constraint independence" [S10, S11], among others, that are also commonly checked.

3.2.6. RQ6: Are the existing approaches automatic (in the sense that user intervention is not required to steer the verification process) and complete?

Automation is closely related to the completeness of the resolution method used to address the reasoning stage, which at the same time is influenced by the degree of support given to OCL. Supporting OCL in its full generality leads to undecidability issues, as stated in the work of Berardi et al. [34], part of [S5]. In this scenario, approaches that are complete are also user-interactive (i.e. not automatic) [S8, S15]. However, the majority of approaches that try to support OCL in its full generality are automatic and ensure termination. This is achieved at the expense of completeness by following bounded verification approaches [S1, S2, S9, S10, S11], in which users typically have to configure beforehand certain parameters to drive the reasoning process, but once it is launched, user intervention is not required. In these cases, results are only conclusive if a solution to the verification problem is found within the boundaries set to the search space. Finally, when only a specific

set of OCL constraints are supported, verification approaches are complete and ensure termination [S4, S6].

3.2.7. RQ7: Are the existing approaches supported by a verification tool?

Those studies not focusing only on the formalization stage, but also on the reasoning stage, are usually complemented by the presence of a verification tool. At the time of this study, the prototypes of [S11, S15, S18] were not available for download, and the same went for AuRUS [S4], although in this last case, a lot of details about the tool are available in [22]. Regarding the rest of tools, in what follows, we briefly comment our experience with them.

UMLtoCSP[13] [S1] is a Java standalone application capable of verifying class diagrams in XMI format created with ArgoUML.[14] The tool supports OCL but constraints must be provided in a separated text file. The GUI looks outdated. It allows loading the model and its constraints, setting the search space boundaries, selecting the correctness properties to be verified and run the verification process. If the verification process succeeds, the tool presents an image of a valid model instance as a proof. The tool requires the ECL^{i}PS^e Constraint Programming System[15] and the graph visualization package Graphviz[16] to work. The last version available (June 2009) presents several bugs and 64-bit platforms or modern operating systems like Windows 7 or Windows 8 are not supported. Besides, it is not unlikely for the verification process to take quite some time in many occasions. The source code is not available on the website.

EMFtoCSP[17] [S2] is an evolution of UMLtoCSP, presented in the form of a set of Eclipse[18] plug-ins. It can be launched through the contextual menu when right-clicking on a model on the "Package Explorer" view. The tool supports EMF models and class diagrams enriched or not with OCL constraints, that can be provided in a separate text file or directly embedded in the model. It offers a wizard-like GUI where users can read OCL constraints from a separate text file, set the boundaries of the search space, select the correctness properties to be verified, and determine the place where to store the results. If the verification process succeeds, the tool is capable of providing a real valid model instance as a proof, and not only and image as its predecessor. As UMLtoCSP, EMFtoCSP requires ECL^{i}PS^e and Graphviz to run. In general, EMFtoCSP corrects many of the limitations and shortcomings of UMLtoCSP, although it also lacks integration with model editing tools. Besides, the wizard-like interface can be an inconvenient when having to verify models multiple times. The last version is from September 2013 and the source code is available for download on the website.

HOL-OCL[19] [S8] is an interactive proof environment that can be used to analyze UML/OCL models. It works on top of the generic proof assistant Isabelle.[20] Unlike the rest of tools analyzed, installing HOL-OCL is not trivial, the tool presenting an important number of prerequisites. HOL-OCL can be used to analyzed models created with ArgoUML although the process is, in general, interactive, and requires building Isabelle theories for that. All in all, HOL-OCL is a tool extremely hard to use for the user not familiarized with formal methods and Isabelle. Last version is 0.9.0 (the year is not indicated) and the downloadable package includes the source code.

Uml2Alloy[21] [S9] is a Java standalone application that can be used to analyze class diagrams enriched with OCL constraints, with the help of Alloy[22] (included with the tool). As UMLtoCSP, the GUI looks outdated. The different steps are distributed in tabs. Once the model is loaded, it is necessary to set the boundaries of the search space and determine how OCL statements will be transformed into Alloy. Finally and after invoking the transformation process explicitly, the analysis of the model can be conducted. If it succeeds, the tool presents a valid model instance as a proof. This instance can then be exported as a PNG image or a PDF document. Overall, we found the tool a bit unintuitive, forcing users to run actions like parsing XMI files or transform the input data to Alloy, explicitly. Last version is 0.52 Beta, built on May 2009. The source code was not available on the website.

USE[23] [S10] is also a Java standalone tool that since version 3.0.5 can be used to verify class diagrams enriched or not with OCL constraints. The tool has a long history behind, since version 0.1 was created in 1999 and, compare to the rest of tools analyzed, it is probably the most polished one. Verification capabilities are provided in the form of a plug-in called "Model Validator"[24] that, once downloaded and uncompressed, must be copied into the "lib/plugins" directory of the tool. USE reads "USE specification files", that is, text files with ".use" extension where the model along with the OCL constraints are described using a particular syntax. USE only verifies satisfiability. Launching the verification process from the GUI requires a text file with information about the boundaries of the different elements in the specification, as well as which OCL constraints must be taken into account during the process. This file is not required, though, if the process is launched from the USE command-line interface, in this case, the tool will generate one with default values. One of the USE nicest features is that the "Model Validator" plug-in supports a catalog of SAT solvers, not only one. If the verification process succeeds, the user must open an "Object diagram" window using the GUI, to see the valid model instance provided as a proof. As typical of these tools, USE does not integrate with modeling editing tools. Last version available is 3.0.6 and the downloadable package includes the source code. The source code for the "Model Validator" plug-in is available in the website.

In general, none of the tools covered here seem to have a strong user base. We delve into what might be some of the reasons for this in Section 5.

3.2.8. RQ8: What type of feedback is obtained from the verification process in each case?

In the majority of cases the feedback provided to inform about the result of the verification process is of the "Yes/No" type, complemented with a sample model in those occasions when the verification process ends successfully. This sample model acts as the proof of the verification results and can be provided in many different formats: as an image (UMLtoCSP [S1]), in XMI format (EMFtoCSP [S2]), as an image or as a PDF file (UML2Alloy [S9]), or as an object diagram (USE [S10]). In addition to this, AuRUS [S4] is capable of yielding some hints in textual form on a pop-up window when the verification process does not succeed (like for example, the list of OCL constraints that cannot be satisfied). This can help model designers to identify the reasons and adjust the model accordingly.

4. Study limitations

After having answered the research questions addressed in this study, now in this section we talk about the limitations of the study, and about those aspects we considered as threats to its validity.

[13] http://gres.uoc.edu/UMLtoCSP/.
[14] http://argouml.tigris.org/.
[15] http://www.eclipseclp.org/.
[16] http://www.graphviz.org/.
[17] http://code.google.com/a/eclipselabs.org/p/emftocsp/.
[18] http://www.eclipse.org/.
[19] http://www.brucker.ch/projects/hol-ocl/.
[20] http://isabelle.in.tum.de/index.html.
[21] http://www.cs.bham.ac.uk/bxb/UML2Alloy/.
[22] http://alloy.mit.edu/alloy/.
[23] http://sourceforge.net/projects/useocl/.
[24] http://sourceforge.net/projects/useocl/files/Plugins/ModelValidator/.

4.1. Study limitations

Admittedly, this study presents certain limitations, that may have affected the retrieval of relevant works. In this regard, we cannot deny the possibility of having missed some relevant papers.

Although nowadays many works are written in English, something that is especially true for those ones aimed at international conferences or journals, it cannot be avoided that limiting the papers considered to those ones written in this language closes the door to the relevant works written in other languages that may exist. Moreover, since in this study we only considered international conferences, workshops and journals, it is also possible that we missed those relevant papers written in English but aimed at national journals and conferences, that may exist.

Another limitation has to do with the repositories discarded. Among these, there are repositories like Google Scholar, Scopus or the ACM Digital Library, which are among the most popular ones, so again, it cannot be denied that their discard may have resulted in missing relevant papers not present in the other repositories we did consider.

However, and in spite of the limitations exposed here, we are confident that we did not miss a large number of relevant works. If any, the amount of works covered in this study makes us think that the conclusions exposed here may be extrapolated to them.

4.2. Threats to validity

As it was the case with the limitations, there are also some threats to the validity of this systematic review. In particular, and according to the way relevant works were retrieved, it might be argued that the combination of structured searches and pruning stages is clearly improvable. Although there are certain issues, as we comment in the next section, that can justify the results, it is also true that when looking back, we identify the following threats.

In the first place, the scope of the conferences, workshops and journals considered does not cover, in general, fields like databases or artificial intelligence (apart from the description logics workshop or the international conference on conceptual modeling), disciplines with a clear relationship with the object of the study. Although we knew in advance about the broad utilization of formalisms based on some type of logic, and the fact that certain studies were clearly rooted in the analysis of entity relationship models or database schemas, we might have failed when deciding not to include more conferences or journals specifically devoted to these fields.

Another issue was the utilization of DBLP as a source for the retrieval of bibliographical references. Although DBLP is a popular resource among computer scientists, the information provided by the bibliographical references gathered from there is limited, and could affect the results obtained in the papers selection stage. In particular, it is typical for the references collected from DBLP to not include keywords. Obviously, this is a limiting feature that might have affected the effectiveness of the keyword pruning stage.

Also related to keywords, it is a fact that their selection is a sensitive issue. Although we conducted the keywords selection process trying to avoid words that are typically used in the field object of this study, it is also true that in those papers using unusual terminology these selection may have been futile. Another strange factor is the evolution of keywords throughout time. We discovered, as we were running our searches over the IEEE Xplore repository, that the keywords assigned to certain entries had been modified. This means that it cannot be discarded the fact that the searches conducted in this review could yield different results in the future affected not only by the appearance of new works, but also because of these changes in the keywords assigned to the entries stored.

The last issue we identified in relation to the validity of our search strategy is the selection of the time span. Although MDE is a relatively young discipline, it is also true that, as mentioned earlier, some of the studies presented here have their roots in fields like databases or artificial intelligence that have been around for much more time than the one covered in this study.

5. Discussion

The main goal of the study presented here was to identify the research work recently done in the field of static model verification, and classify it according to a number of research questions posed in Section 1. In addition to this, in this section we discuss a number of findings we discovered while doing this study that, in our opinion, are worth noting.

5.1. Terminology

The first issue we discovered was the absence of a precise and rigorous terminology, shared among all the verification approaches analyzed. One implication of this is the difficulty for contextualizing the works in this field, especially when compared with works from other fields. In this regard, the main challenge we had was regarding the works defining themselves as works addressing "model consistency checking". In fact, this study has reinforced our initial impression of the existence of a gray zone when trying to determine what the exact boundaries are between model verification and model consistency checking (hence, the inclusion of search terms like "consistency" or "checking" in the search strings shown in Table 10). Some works use the term consistency checking to refer to inter-model relationships that must hold (e.g. a call to a method of a class in a UML sequence diagram should require that the same class in the UML class diagram includes the corresponding method defined). These works clearly fall outside of the scope of works analyzed there. Nevertheless, other works use the term consistency checking in a broader sense overlapping with what we defined here as satisfiability, see for instance the different notions of consistency introduced by Wahler et al. [72]. From that point of view, all the works covered here could also be regarded as consistency checking approaches.

Clearly, an unambiguous definition of all the words around the concept of model verification (including verification itself, validation, consistency, well-formedness and so forth) is needed.

5.2. Correctness properties

The lack of a precise and rigorous terminology affects also the way correctness properties are named and defined. One example of this is the correctness property commonly referred to as "satisfiability", the most popular one among the verification approaches object of this study (hence, the inclusion of the term "satisfiability" in the search strings shown in Table 10). After having read the papers collected for the study, we realized that there were at least 6 different ways of referencing satisfiability. In some cases, this lack of homogeneity might be understandable. After all, "satisfiability" is a term widely used so it should be normal that different papers used different notions of the word (e.g. in some satisfiability may be what others call strong satisfiability while others may use the notion of weak satisfiability; with both concepts in its turn lacking also a precise and unambiguous definition). Unfortunately, the differences in meaning were not always so slightly different and sometimes different flavors of satisfiability coexisted even in the same approach. All these situations are problematic since when

the reader has a preconceived definition for the property in mind, and this is not the same as the one used in the paper, this will likely induce to errors in the interpretation of the text. Furthermore, and to make matters worse, certain correctness properties could be expressed in terms of others more "core" ones (e.g. constraint redundancy can be expressed in terms of constraint subsumption which in turn can be expressed in terms of a satisfiability relationship).

In our opinion, this lack of homogeneity when precisely naming and defining correctness properties could be clearly improved with the creation of a catalog of correctness properties, where to find the list of correctness properties that can be checked when addressing formal verification of static models. In this catalog, precise and unambiguous names and meanings should be given to the different correctness properties, as well as a clear description of how they are related to each other. As far as we know, no efforts have been made so far in this direction.

5.3. Lack of adequate verification tools

Another important finding is related to the adequacy of existing verification tools. Although at first sight, by looking at Table 20 the situation may not look so bad, the reality is that the number of verification tools available is certainly limited. Among the tools listed in Table 20, the tool EMFtoCSP [S2] is an evolution of a previous one called UMLtoCSP [S1] and others were created with the intent of addressing validation issues, like USE [S10] or UML2Alloy [S9]. If we add to this that tools like HOL-OCL [S8] require from the user a non-negligible expertise on formal aspects, we may conclude that the existing offer of verification tools, apart from being certainly limited in size, is in some cases targeted at a very limited audience.

In our opinion, a verification tool, in order to be effective and widely adopted, has to present, at least, four important characteristics: first, it should hide all the complexities derived from the utilization of formal methods, up to the point of making their presence transparent to the end user. Second, it should integrate seamlessly into the model designer tool chain. Third, it should provide a meaningful feedback. And four, it should be reasonably efficient, not making users to wait for ages when verifying large, real-world models. We believe these aspects are, from an end-user point of view, more important than other more formal aspects, like the completeness of the results.

Unfortunately, none of the verification tools analyzed in this study does a good job at fulfilling all these requirements. In general, these tools do not integrate well, and have been designed to conduct the verification separately from the rest of tasks that characterize the work of a model designer. When it comes to hiding the underlying formal methods employed, the situation is better, especially in the case of bounded verification approaches, although having to manually set the boundaries of the search space (as it is the case, for example, in [S1], [S2] and [S9]) can be an issue when verifying large models. The feedback can be considered acceptable when the model under analysis is found to be correct, but is clearly insufficient in the other case, with the majority of tools yielding no feedback on where to apply fixes if the model is found to be not valid (to the best of our knowledge, only [S4] provides some hints to help users on this). Finally, efficiency is a major issue. In general these tools behave well when dealing with toy examples or models of reduced size, but the performance drops dramatically when they are used to verify large models or real-world examples.

All in all, it comes as no surprise that none of these tools seem to have a strong user base. At this moment, model verification can be regarded as an unpleasant experience, that forces users to switch back and forth between model editors and verification tools to check for errors every time models are refined, usually with little or no clue on where to apply fixes if the verification fails.

We would not like to finish this section, though, without contributing some ideas on how these deficiencies could be addressed. Regarding integration with other tools and hiding complexity, a major effort on development tasks is clearly needed. Improving efficiency and feedback is clearly related to the underlying solvers employed during the verification process. These tools have experienced a dramatic improvement in the last few years, but still, even more improvements are needed. Meanwhile, doing research on techniques to cleverly define the search space boundaries of bounded verification approaches, and on incremental verification techniques, could alleviate this.

5.4. Difficulties to compare verification tools

Another finding, also related to verification tools, is the difficulty to evaluate and compare the coverage and performance of the tools.

The majority of tools analyzed tend to be accompanied by a set of samples (a small number of input models where to check the correctness properties covered by the tool), that are usually simplistic and not representative of real scenarios. Although the existence of sample input models is always welcome, and their simplicity can be linked to the space limitations in research papers, this limits the performance analysis of the tools. And since the samples obviously vary from one tool to another, running comparisons between different verification tools is considerably more complex. Interoperability problems between modeling tools and differences on the modeling constructs each tool supports (and the terminology they use as discussed before) complicate even more the situation.

We think a possible way to improve the current situation would be the creation of a standard set of benchmarks as typically done in other communities (e.g. see the TPC transaction processing database benchmarks[25]). These benchmarks, which must be based on the catalog of correctness properties proposed before, should be composed of multiple sets of models of varying sizes and complexity, accompanied by the list of correctness properties that could be checked on them, as well as the expected results. The existence of these benchmarks could not only facilitate running performance analysis among different verification tools, but also enhance their development, since they could also be used to test the tools. As of now, we are not aware of the existence of initiatives regarding the creation of benchmarks for model verification tools.

5.5. The importance of the snowballing process

The last finding we would like to comment is not related with the objective of this study, but with the systematic procedure we followed to retrieve the set of related works. Admittedly speaking, when we saw that the number of papers collected out of the combination of structured searches and pruning stages was only 22, our first feeling was of slight disappointment. This disappointment was originally caused by the fact that, by the time when we were considering the possibility of running this study, we had thought that the number of existing works devoted to verification of static models was actually bigger than the number of results retrieved. However, this initial impression was quickly replaced by a combination of mixed feelings when, throughout the process of snowballing, we gather 26 additional works. On the one hand, it was satisfactory to see that our original assumption on the number of existing works was not completely wrong, but on the other hand, we started to think that something had gone wrong with the structural searches we had run. Before conducting our study, we

[25] http://www.tpc.org/.

had reviewed many systematic literature reviews in which the majority of relevant works, if not all of them, were gathered by means of structural searches, causing snowballing processes to yield little or no results at all. Obviously, since in our case the process of snowballing had played a very important role during the retrieval of relevant works, we started to wonder what could have gone wrong with our structured searches. Although, as we mentioned in Section 4, it cannot be discarded that by including additional sources or using a different set of search strings, the number of results gathered out of structured searches and pruning stages would have been higher, we discovered two fundamental aspects that made us think that trying to collect the majority of relevant papers just by running structured searches would have turned the process into an unmanageable one. The first aspect was that when looking at where the relevant papers came from, there was a considerable diversity in the sources, and the second, which has been already addressed in the first two findings commented here, was the lack of a precise naming convention, not only with respect to the terms most commonly used in model verification, but also in relation to what the boundaries of model verification are, especially when compared to model consistency checking.

Our impression is that in certain situations, in which factors like the ones commented here arise, the utilization of other techniques like snowballing processes, rather to solely or primarily focus on structured searches, is key, since it makes possible the retrieval of works without spending a lot of time in designing too many structured searches that cannot guarantee the coverage of a substantial part of the whole set of relevant works. Moreover, in the particular case of this study, although the structured searches did not cover the whole set of works, they were clearly useful since they allowed us to identify an important set of core papers, from which running the process of snowballing. Therefore, the structured searches were important not only to retrieve an important number of works, but also to facilitate the execution of the snowballing process. This has made us think that an adequate combination of structured searches and snowballing can sometimes produce the best results.

6. Conclusions and outlook

In this paper we have presented a systematic literature review on the formal verification of static software models. Throughout a combination of systematic searches, systematic pruning of non-relevant papers and a comprehensive snowballing process, we identified 48 relevant works that were grouped into 18 different studies for their analysis.

We analyzed these 18 studies to answer the following research questions: (RQ1) What are the typical phases in model verification approaches? (RQ2) What are the formal methods and techniques employed in the verification approaches? (RQ3) What are the types of static models that can be verified in each case? (RQ4) Up to what extent is the Object Constraint Language (OCL) supported? (RQ5) Which correctness properties are verified in each case? (RQ6) Are the existing approaches automatic (in the sense that user intervention is not required to steer the verification process) and complete? (RQ7) Are the existing approaches supported by a verification tool? and (RQ8) What type of feedback is obtained from the verification process in each case?

The analysis conducted indicates that the existing verification approaches are based on an initial stage of formalization which then gives way to a reasoning phase in which verification is conducted using the mechanisms available in the target formalism, although not all the studies cover this second stage. The formalization is based on the utilization of some sort of logical or mathematical representation of the models, usually expressed in the form of UML class diagrams or similar, complemented with OCL constraints (not always covered in its full generality by the studies we analyzed). Regarding the reasoning stage, the most typically verified property is that of satisfiability with some approaches aimed at checking the relationship between constraints (like "constraint redundancy") as well. Some studies also provide a verification tool implementing the techniques described in the study. The completeness of such tools is strongly influenced by the degree of support given to OCL. In this regard, the majority of tools aiming at giving full OCL support follow a bounded verification approach, in order to keep the verification process automatic and ensure its termination, although at the expense of its completeness. Feedback is mostly limited to a "Yes/No" result complemented with a sample model in those cases where the verification process succeeds.

We hope this systematic review helps the community to better understand the current situation regarding the formal verification of static software models and what is needed to move forward. In our opinion, the first challenges that should be addressed are the catalog of verification properties, the agreement on a shared and precise terminology and the definition of benchmarks to be able to evaluate and compare the approaches and tools available. Regarding this last aspect, we would like to emphasize that two key aspects to improve the adoption of verification tools by end-users are first, making them more efficient, and second, improving their usability, especially when it comes to their seamless integration in the model designer tool chain, and the feedback provided when the verification fails. We believe these improvements would clearly enhance the landscape in the field of formal verification of static models, and would facilitate the appearance of a new generation of better verification tools ready for mainstream adoption.

References

[1] C.B. Jones, P.W. O'Hearn, J. Woodcock, Verified software: a grand challenge, IEEE Comp. 39 (4) (2006) 93–95.
[2] C.A. González, F. Büttner, R. Clarisó, J. Cabot, EMFtoCSP: a tool for the lightweight verification of EMF models, in: Proceedings of Formal Methods in Software Engineering: Rigorous and Agile Approaches, FormSERA 2012, 2012, pp. 44–50.
[3] P. Mohagheghi, V. Dehlen, T. Neple, Definitions and approaches to model quality in model-based software development – a review of literature, Inform. Softw. Technol. 51 (12) (2009) 1646–1669.
[4] F.J. Lucas, F. Molina, J.A.T. Álvarez, A systematic review of UML model consistency management, Inform. Softw. Technol. 51 (12) (2009) 1631–1645.
[5] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Tech. Rep. EBSE 2007-001, Keele University and Durham University, 2007.
[6] B. Kitchenham, Procedures for Performing Systematic Reviews, Tech. rep., Departament of Computer Science, Keele University, 2004.
[7] E.M.C. Jr., O. Grumberg, D.A. Peled, Model Checking, The MIT Press, 1999.
[8] A. Artale, D. Calvanese, Y.A. Ibá nez-García, Checking full satisfiability of conceptual models, in: 23rd International Workshop on Description Logics, DL 2010, CEUR Workshop Proceedings, vol. 573, CEUR-WS.org, 2010.
[9] A. Artale, D. Calvanese, Y.A. Ibá nez-García, Full satisfiability of UML class diagrams, in: 29th International Conference on Conceptual Modeling, ER 2010, Lecture Notes in Computer Science, vol. 6412, Springer, 2010, pp. 317–331.
[10] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, M. Zakharyaschev, Complexity of reasoning in entity relationship models, in: 20th International Workshop on Description Logics, DL 2007, CEUR Workshop Proceedings, vol. 250, CEUR-WS.org, 2007.
[11] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, M. Zakharyaschev, Reasoning over extended ER models, in: 26th International Conference on Conceptual Modeling, ER 2007, Lecture Notes in Computer Science, vol. 4801, Springer, 2007, pp. 277–292.
[12] M. Balaban, A. Maraee, A UML-Based method for deciding finite satisfiability in description logics, in: 21st International Workshop on Description Logics, DL 2008, CEUR Workshop Proceedings, vol. 353, CEUR-WS.org, 2008.
[13] D. Berardi, D. Calvanese, G.D. Giacomo, Reasoning on UML class diagrams is EXPTIME-hard, in: 2003 International Workshop on Description Logics, DL 2003, CEUR Workshop Proceedings, vol. 81, CEUR-WS.org, 2003.
[14] J. Cabot, R. Clarisó, D. Riera, UMLtoCSP: A tool for the formal verification of UML/OCL models using constraint programming, in: 22nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2007, 2007, pp. 547–548.
[15] J. Cabot, R. Clarisó, D. Riera, Verification of UML/OCL class diagrams using constraint programming, in: Proceedings of the Workshops of the 1st

International Conference on Software Testing Verification and Validation, ICST 2008, IEEE Computer Society, 2008, pp. 73–80.
[16] F. Büttner, J. Cabot, Lightweight string reasoning for OCL, in: 8th European Conference on Modelling Foundations and Applications, ECMFA 2012, Lecture Notes in Computer Science, vol. 7349, Springer, 2012, pp. 244–258.
[17] M. Cadoli, D. Calvanese, T. Mancini, Finite satisfiability of UML class diagrams by constraint programming, in: 2004 International Workshop on Description Logics, DL 2004, CEUR Workshop Proceedings, vol. 104, CEUR-WS.org, 2004.
[18] M. Clavel, M. Egea, M.A.G. de Dios, Checking unsatisfiability for OCL constraints, Electronic Communication of the European Association of Software Science and Technology (ECEASST), 24.
[19] M. Gogolla, F. Büttner, M. Richters, USE: a UML-based specification environment for validating UML and OCL, Sci. Comp. Program. 69 (1–3) (2007) 27–34.
[20] A. Maraee, M. Balaban, Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets, in: 3rd European Conference on Model Driven Architecture-Foundations and Applications, ECMDA-FA 2007, Lecture Notes in Computer Science, vol. 4530, Springer, 2007, pp. 17–31.
[21] A. Queralt, A. Artale, D. Calvanese, E. Teniente, OCL-Lite: finite reasoning on UML/OCL conceptual schemas, Data Knowl. Eng. 73 (2012) 1–22.
[22] A. Queralt, G. Rull, E. Teniente, C. Farré, T. Urpí, AuRUS: automated reasoning on UML/OCL schemas, in: 29th International Conference on Conceptual Modeling, ER 2010, Lecture Notes in Computer Science, vol. 6412, Springer, 2010, pp. 438–444.
[23] A. Queralt, E. Teniente, Reasoning on UML class diagrams with OCL constraints, in: 25th International Conference on Conceptual Modeling, ER 2006, Lecture Notes in Computer Science, vol. 4215, Springer, 2006, pp. 497–512.
[24] A. Queralt, E. Teniente, Decidable reasoning in UML schemas with constraints, in: 20th International Conference on Advanced Information Systems Engineering, CAiSE 2008, Lecture Notes in Computer Science, vol. 5074, Springer, 2008, pp. 281–295.
[25] A. Queralt, E. Teniente, Verification and validation of UML conceptual schemas with OCL constraints, ACM Trans. Softw. Eng. Method. 21 (2) (2012) 13:1–13:41.
[26] L.A. Rahim, Mapping from OCL/UML metamodel to PVS metamodel, International Symposium on Information Technology, ITSim 2008, vol. 1, IEEE Computer Society, 2008, pp. 1–8.
[27] A. Shaikh, R. Clarisó, U.K. Wiil, N. Memon, Verification-driven slicing of UML/OCL models, in: 25th IEEE/ACM International Conference on Automated Software Engineering, ASE 2010, ACM, 2010, pp. 185–194.
[28] M. Szlenk, Formal semantics and reasoning about UML class diagram, in: 1st International Conference on Dependability of Computer Systems, DepCoS-RELCOMEX 2006, IEEE Computer Society, 2006, pp. 51–59.
[29] T. Ali, M. Nauman, M. Alam, An accessible formal specification of the UML and OCL meta-model in Isabelle/HOL, in: 2007 IEEE International Multitopic Conference, INMIC 2007, IEEE Computer Society, 2007, pp. 1–6.
[30] K. Anastasakis, B. Bordbar, G. Georg, I. Ray, On challenges of model transformation from UML to Alloy, Softw. Syst. Model. 9 (1) (2010) 69–86.
[31] K. Anastasakis, B. Bordbar, G. Georg, I. Ray, UML2Alloy: a challenging model transformation, in: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2007, Lecture Notes in Computer Science, vol. 4735, Springer, 2007, pp. 436–450.
[32] B. Beckert, U. Keller, P.H. Schmitt, Translating the object constraint language into First-Order predicate logic, in: 2nd Verification Workshop, VERIFY 2002, DIKU Technical Reports, vol. 2002/07, Department of Computer Science, University of Copenhagen, DIKU, 2002, pp. 113–123.
[33] D. Berardi, Using DLs to reason on UML class diagrams, in: Workshop on Applications of Description Logics, ADL-2002, CEUR Workshop Proceedings, vol. 63, CEUR-WS.org, 2002.
[34] D. Berardi, D. Calvanese, G.D. Giacomo, Reasoning on UML class diagrams, Artif. Intell. 168 (1–2) (2005) 70–118.
[35] A.D. Brucker, J. Doser, B. Wolff, An MDA framework supporting OCL, Electronic Communication of the European Association of Software Science and Technology (ECEASST), 5.
[36] A.D. Brucker, B. Wolff, The HOL-OCL book, Tech. Rep. 525, ETH Zurich, 2006.
[37] A.D. Brucker, B. Wolff, HOL-OCL: a formal proof environment for UML/OCL, in: 11th International Conference on Fundamental Approaches to Software Engineering, FASE 2008, Lecture Notes in Computer Science, vol. 4961, Springer, 2008, pp. 97–100.
[38] M. Cadoli, D. Calvanese, G.D. Giacomo, Towards implementing finite model reasoning in description logics, in: 2004 International Workshop on Description Logics, DL 2004, CEUR Workshop Proceedings, vol. 104, CEUR-WS.org, 2004.
[39] M. Cadoli, D. Calvanese, G.D. Giacomo, T. Mancini, Finite model reasoning on UML class diagrams via constraint programming, in: 10th Congress of the Italian Association for Artificial Intelligence and Human-Oriented Computing, AI*IA 2007, Lecture Notes in Computer Science, vol. 4733, Springer, 2007, pp. 36–47.
[40] A. Calì, D. Calvanese, G.D. Giacomo, M. Lenzerini, A formal framework for reasoning on UML class diagrams, in: 13th International Symposium Foundations of Intelligent Systems, ISMIS 2002, Lecture Notes in Computer Science, vol. 2366, Springer, 2002, pp. 503–513.
[41] A. Formica, Satisfiability of object-oriented database constraints with set and bag attributes, Inform. Syst. J. 28 (3) (2003) 213–224.
[42] A. Formica, Finite satisfiability of integrity constraints in Object-Oriented database schemas, IEEE Trans. Knowl. Data Eng. 14 (1) (2002) 123–139.
[43] M. Gogolla, J. Bohling, M. Richters, Validating UML and OCL models in USE by automatic snapshot generation, Softw. Syst. Model. 4 (4) (2005) 386–398.
[44] M. Gogolla, L. Hamann, M. Kuhlmann, Proving and visualizing OCL invariant independence by automatically generated test cases, in: 4th International Conference on Tests and Proofs, TAP 2010, Lecture Notes in Computer Science, vol. 6143, Springer, 2010, pp. 38–54.
[45] M. Gogolla, M. Kuhlmann, L. Hamann, Consistency independence and consequences in UML and OCL models, in: 3rd International Conference on Tests and Proofs, TAP 2009, Lecture Notes in Computer Science, vol. 5668, Springer, 2009, pp. 90–104.
[46] R.M. Kamenoff, N. Lévy, Using B formal specifications for analysis and verification of UML/OCL models, in: Workshop on Consistency Problems in UML-based Software Development, 5th International Conference on the Unified Modeling Language, 2002.
[47] M. Kuhlmann, M. Gogolla, From UML and OCL to relational logic and back, in: ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems, MODELS 2012, Lecture Notes in Computer Science, vol. 7590, Springer, 2012, pp. 415–431.
[48] M. Kuhlmann, M. Gogolla, Strengthening SAT-Based validation of UML/OCL models by representing collections as relations, in: 8th European Conference on Modelling Foundations and Applications, ECMFA 2012, Lecture Notes in Computer Science, vol. 7349, Springer, 2012, pp. 32–48.
[49] M. Kuhlmann, L. Hamann, M. Gogolla, Extensive validation of OCL models by integrating SAT solving into USE, in: 49th International Conference on Objects, Models, Components and Patterns, TOOLS 2011, Lecture Notes in Computer Science, vol. 6705, Springer, 2011, pp. 290–306.
[50] A. Queralt, A. Artale, D. Calvanese, E. Teniente, OCL-Lite: a decidable (yet expressive) fragment of OCL, in: 25th International Workshop on Description Logics, DL 2012, CEUR Workshop Proceedings, vol. 846, CEUR-WS.org, 2012.
[51] D. Roe, K. Broda, A. Russo, R. Russo, Mapping UML Models Incorporating OCL Constraints into Object-Z, Tech. rep., Department of Computer Science, Imperial College London, 2003.
[52] G. Rull, C. Farré, E. Teniente, T. Urpí, Providing explanations for database schema validation, in: 19th International Conference on Database and Expert Systems Applications, DEXA 2008, Lecture Notes in Computer Science, vol. 5181, Springer, 2008, pp. 660–667.
[53] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, R. Drechsler, Verifying UML/OCL models using boolean satisfiability, in: Design, Automation and Test in Europe, DATE 2010, IEEE, 2010, pp. 1341–1344.
[54] M. Soeken, R. Wille, R. Drechsler, Encoding OCL data types for SAT-Based verification of UML/OCL models, in: 5th International Conference on Tests and Proofs, TAP 2011, Lecture Notes in Computer Science, vol. 6706, Springer, 2011, pp. 152–170.
[55] M. Lenzerini, P. Nobili, On the satisfiability of dependency constraints in entity-relationship schemata, Inform. Syst. J. 15 (4) (1990) 453–461.
[56] C. Farré, E. Teniente, T. Urpí, Checking query containment with the CQC method, Data Knowl. Eng. 53 (2) (2005) 163–223.
[57] C. Farré, G. Rull, E. Teniente, T. Urpí, SVTe: a tool to validate database schemas giving explanations, in: 1st International Workshop on Testing Database Systems, DBTest 2008, ACM, 2008, p. 9.
[58] D. Calvanese, Finite model reasoning in description logics, in: 5th International Conference on Principles of Knowledge Representation and Reasoning, KR 96, Morgan Kaufman, 1996, pp. 292–303.
[59] A. Formica, M. Missikoff, Integrity constraints representation in object-oriented databases, in: 1st International Conference on Information and Knowledge Management: Expanding the Definition of "Database", CIKM 92, Selected Papers, Lecture Notes in Computer Science, vol. 752, Springer, 1993, pp. 69–85.
[60] C. Beeri, A formal approach to object-oriented databases, Data Knowl. Eng. 5 (1990) 353–382.
[61] T. Nipkow, L.C. Paulson, M. Wenzel, Isabelle/HOL: a proof assistant for higher-order logic, Lecture Notes in Computer Science, vol. 2283, Springer, 2002.
[62] D. Jackson, Software Abstractions: Logic, Language, and Analysis, The MIT Press, 2006.
[63] E. Torlak, D. Jackson, Kodkod: A relational model finder, in: 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007, Vol. 4424 of Lecture Notes in Computer Science, Springer, 2007, pp. 632–647.
[64] S. Owre, N. Shankar, J.M. Rushby, D.W.J. Stringer-Calvert, PVS Language Reference, SRI International, Computer Science Laboratory, 1999.
[65] D.S. Kolovos, R.F. Paige, F.A.C. Polack, Eclipse development tools for Epsilon, in: Eclipse Summit Europe, Eclipse Modeling Symposium, 2006.
[66] B. Beckert, R. Hähnle, P. Schmitt (Eds.), Verification of object-oriented software: the KeY approach, Lecture Notes in Computer Science, vol. 4334, Springer, 2006.
[67] R. Duke, G. Rose, G. Smith, Object-Z: a specification language advocated for the description of standards, Comp. Stand. Interf. 17 (5-6) (1995) 511–533.
[68] G. Smith, The Object-Z Specification Language (Advances in Formal Methods), Springer, 1999.
[69] J.M. Spivey, The Z Notation: A Reference Manual, Prentice-Hall, 1989.

[70] S.-K. Kim, D. Carrington, A formal mapping between UML models and Object-Z specifications, in: ZB 2000: Formal Specification and Development in Z and B, 1st International Conference of B and Z Users, Lecture Notes in Computer Science, vol. 1878, Springer, 2000, pp. 2–21.

[71] J.R. Abrial, The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[72] M. Wahler, D.A. Basin, A.D. Brucker, J. Koehler, Efficient analysis of pattern-based constraint specifications, Softw. Syst. Model. 9 (2) (2010) 225–255.