Information and Software Technology 51 (2009) 1631–1645

Contents lists available at ScienceDirect

Information and Software Technology

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

A systematic review of UML model consistency management [☆]

Francisco J. Lucas [*], Fernando Molina [1], Ambrosio Toval

Software Engineering Research Group, Department of Informatics and Systems, University of Murcia, Spain

ARTICLE INFO

Article history:
Available online 3 May 2009

Keywords:
UML
Model consistency
Systematic literature review

ABSTRACT

Information System (IS) development has been beset by consistency problems since its infancy. These problems are greater still in UML software development, and are principally caused by the existence of multiple views (models) for the same system, and may involve potentially contradictory system specifications. Since a considerable amount of work takes place within the scope of model consistency management, this paper presents a systematic literature review (SLR) which was carried out to discover the various current model consistency conceptions, proposals, problems and solutions provided. To do this, a total of 907 papers related to UML model consistency published in literature and extracted from the most relevant scientific sources (IEEE Computer Society, ACM Digital Library, Google Scholar, ScienceDirect, and the SCOPUS Database) were considered, of which 42 papers were eventually analyzed. This systematic literature review resulted in the identification of the current state-of-the-art with regard to UML model consistency management research along with open issues, trends and future research within this scope. A formal approach for the handling of inconsistency problems which fulfils the identified limitations is also briefly presented.

1. Introduction

Consistency problems have existed in Information System (IS) development since its beginning and are usually linked to the existence of multiple models or views which participate in the development process.

Problems related to the maintenance and management of consistency were identified in the design of the first databases. These problems were, on some occasions, due to the duplication of information as a result of design errors and, on others, caused by the need to denormalize this design in order to obtain a more efficient database execution.

When focusing on software development using UML (Unified Modelling Language) [1], problems related to consistency between models also appear. This is mainly due to the existence of multiple views (models) for the same system, which may potentially contain contradictory specifications. These inconsistencies among different models or views of a system may be a source of numerous errors in the software developed [2] and may, moreover, consequently complicate its management [3]. A proof of the importance of UML model consistency in software development is the existence of both a considerable amount of work within this scope (see Section 3.6), and specific workshops and conference sessions attempting to deal with this kind of problems.

Furthermore, in recent years, the profound impact of the Model Driven Engineering (MDE) [4] and particularly of the Model Driven Architecture (MDA) proposal [5], in which the use of models guides the development of a system, has caused models to gain more importance. According to [6], inconsistency problems have become critical within the scope of MDE and could make the use of models as a source of automatic code generation impossible.

Attempts to solve these problems have led to the appearance of a great deal of work in model consistency management. This paper presents a systematic literature review (SLR), which was carried out in order to discover the various current consistency conceptions, proposals, problems and solutions provided. A systematic literature review provides a means of identifying, evaluating, and interpreting the literature relevant to a particular research question or topic area [7]. There are numerous reasons for carrying out systematic literature reviews such as:

- To summarize the existing evidence concerning a treatment or technology, e.g. to summarize the empirical evidence of the benefits and limitations of a specific agile method.
- To identify any gaps in current research in order to suggest areas for further investigation.
- To provide a framework/background in order to appropriately position new research activities.

[☆] Partially financed by the Spanish Ministry of Science and Technology, project DEDALO TIN2006-15175-C05-03.

[*] Corresponding author.

E-mail addresses: fjlucas@um.es (F.J. Lucas), fmolina@um.es (F. Molina), atoval@um.es (A. Toval).

[1] Supported by the Fundación Séneca (Región de Murcia).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.04.009

This SLR has been carried out by considering a total of 907 papers related to UML published in literature and extracted from the most relevant scientific sources (IEEE Computer Society, ACM Digital Library, Google Scholar, ScienceDirect, and the SCOPUS Database). The selected papers have been classified according to a number of features within the scope of model consistency management. In the context of this report, this systematic literature review attempts to answer the following question: What inconsistency problems have been tackled by the current approaches for model consistency management and how extensible are they in guaranteeing its usability and maintainability, allowing the addition of new problems and types of models? As a result of this systematic literature review, the present state of consistency management research is identified, along with open issues, trends and future research within this scope. Moreover, a formal approach with which to handle inconsistency problems based on transformation languages, algebraic specifications and rewriting logic [8] which will overcome the identified limitations is briefly presented.

Another survey on inconsistency management [9], which was carried out in 2001, focused on the activities of an inconsistency management process, and the positive features and limitations of the approaches that can be used in each activity. However, although the main limitations of each approach studied have also been included in our review process (see [10]), the main aim of this SLR is different since it focuses on the type of problems tackled by current work, the diagrams supported and the maintainability of each proposal, in order to answer the research question that has motivated the SLR.

The remainder of the paper is structured as follows: Section 2 describes the background and the context of this work, and the systematic literature review developed. Section 3 shows what an SLR is and describes how that which appears in this paper has been carried out. The results of the SLR are analyzed in Section 4. Finally, a preliminary approach with which to overcome the limitations found in the SLR is shown, along with our conclusions.

2. Background: model consistency concepts

Several possible definitions of model consistency and its classification appear in existing literature. This is sometimes due to the fact that these concepts are used in various, and even ambiguous or contradictory, ways within different contexts [11]. The following definitions and sources adopted for each concept related to consistency have been used in order to unify the terminology used in this paper:

- Consistency. A state in which two or more elements, which overlap in different models of the same system, have a satisfactory joint description [9].
- Inconsistency Management. A set of activities for detecting and handling consistency problems.
- Horizontal or intra-model consistency problems. These are problems between models which are built at the same level of modelling abstraction [12].
- Vertical or inter-model consistency problems. These problems can appear between models built at different levels of abstraction, for example, when refinements are made during the development [12].
- Syntactic consistency problems. This kind of consistency should guarantee that a model conforms to its abstract syntax (specified by its metamodel) [13].
- Semantic consistency problems. This consistency requires that models' behaviour be semantically compatible [13].

These definitions will be used in the remainder of the paper. The classifications between horizontal–vertical and syntactic–semantic consistency problems are orthogonal. Table 1 shows an example of each of the main classifications described (horizontal–vertical, syntactic–semantic). All the papers analyzed are included in one of these four categories.

Inconsistency problems may originate from various sources. [12] identifies two main sources of these problems:

- The multiview nature of the models: a system is described as multiple views that give the details of different concerns that make up the system, with a possible overlap between them.
- The system is developed through multiple phases or iterations, and each one produces a new, more refined description of the system.

In addition to these sources, the distributed development of a system with potentially multiple developers, which may on occasions be geographically distributed (local or global development), with multiple interpretations of both requirements and the UML notation itself [12] may also produce inconsistency problems.

3. Systematic literature review

This section shows a summary of the SLR presented in this paper. The complete report of this SLR can be found in [10].

The process of conducting a systematic literature review is not a simple task, since the whole process must be precisely defined and documented, and must include intermediate results. To facilitate the planning and execution of systematic literature reviews, [14] proposes a review protocol template based on a systematic literature review protocol previously developed in the area of medicine, along with other guidelines developed in the area of Software Engineering. In this paper, we have carried out the SLR by using the aforementioned template, and have also taken into account the guidelines given by [7]. Therefore, this template has been slightly extended to include some of the sections mentioned in [7], which were not explicitly included.

The SLR presented was carried out by effectuating the following activities:

1. Question formularization.
2. Source selection.
3. Studies selection process.
4. Information extraction.
5. Extraction execution.

The sections below give a detailed explanation of how each of these activities was carried out.

3.1. Question formularization

In this paper, and by following the instructions given in [7,14], we have defined the following research question in order to guide the SLR:

- What inconsistency problems have been tackled by the current approaches for model consistency management and how extensible are they in guaranteeing its usability and maintainability, allowing the addition of new problems and types of models?

This main question has been refined into the following set of more specific research questions (RQ):

(RQ1) What is the UML version used in the work?

Table 1
Concrete examples of consistency problem classifications.

Type of consistency	Syntactic	Semantic
Horizontal	The class names used in a sequence diagram should appear in the associated class diagram	The events produced in a sequence diagram should not produce inconsistent states in the state diagrams of the objects which participate in the interaction
Vertical	The methods definition of a class should be consistent in all the abstraction levels in which these methods could be defined	When some child classes are created in a refinement from a parent class, the traces defined by the statemachine of the parent class should be supported by the low-level statemachines of the child classes

(RQ2) What is the method, language or technique used to carry out the approach (and is it a formal or a non-formal technique)? This distinction is made due to the fact that formal techniques have some features which help to avoid consistency problems such as the imprecise or ambiguous interpretation of the semantics of the modelling language [12], and also offer a wide range of applications based on their mathematical underpinning. A technique is considered to be a formal technique if it specifies at meta-level (a) a syntax, (b) semantics, and (c) a proof system [15].

(RQ3) What types of diagrams have been tackled in each approach?

(RQ4) What kind of consistency problems have been tackled in each approach?

(RQ5) How can the approach be extended to support new consistency checks? This question is related to both the usability and the maintainability of the proposal. The most up to date solutions for these problems are only partial and are of an academic nature [6], partly due to the impossibility of implementing all the possible checks by the authors of each approach. Thus, if a modeller wishes to include new consistency checks, s/he has to specify or implement them directly over the language or technique used in the approach, which may often be unknown to the modellers. The way in which each approach can be extended will therefore be considered as a further research question.

(RQ6) How suitable is the integration of the approach within CASE tools? Another feature that an approach should offer for its use within an industrial software development environment is that of good support within a CASE tool, since this helps to provide suitable feedback during modelling. In order to obtain a suitable integration between approach and tool, on the one hand, the approach should be easy for the modellers to use as a result of its integration within the CASE tool and, on the other hand, the extension mechanism offered by the approach to express new problems should also be integrated within the CASE tool. This will be considered in each work.

(a) For those approaches that do not offer integration within a CASE tool, we will consider whether they offer a prototype tool through which to facilitate the use, or learning, of the approach.

After carrying out this SLR we expect to obtain:

- Relevant information concerning the present state-of-the-art with regard to UML model consistency management.
- The identification of gaps in current research, solutions, trends and future research within this scope.
- Recommendations concerning the features that should be included in an approach which offers suitable and extensible support in the solution of consistency management problems. These will be extracted from the conclusions of the SLR.

As a consequence of these results, the definition of a novel preliminary approach for handling inconsistency problems which fulfils these recommendations is briefly presented in Section 5.

3.2. Source selection

The sources identified for use in the search for primary studies were those recommended by [7] which, from our point of view, were also appropriate for this review since they contain the work published in those journals, conferences and workshops which are of recognized quality within the research community. These sources are:

- IEEE Computer Society, search fields: title, abstract and full text;
- ACM Digital Library, search fields: title, abstract and review;
- Google Scholar, search fields: title, abstract and full text;
- Science direct, search fields: title, abstract and full text; and
- The SCOPUS Database, search fields: title, abstract and keywords.

Other important sources such as DBLP or CiteSeer were not explicitly included since they were indexed by some of the mentioned sources (e.g., Google Scholar and SCOPUS Database). In the selected sources, we experimented with various search string criteria. That which eventually retrieved the highest number of useful results was:

(“management” AND “model” AND (“inconsistency” OR “consistency”))

OR (“model” AND (“inconsistency” OR “consistency”))

Certain synonyms and terms related to the concept of the model within the scope of consistency management were also taken into account in the search process. Specifically, the terms diagram, view and concern have been used as synonyms of model.

3.3. Studies selection process

Having defined the source selection, we shall now describe the process used to identify those studies that provided direct evidence with regard to the research question. We shall do this by defining a basic inclusion and exclusion criteria based on the research question, along with a procedure through which this selection was made. This is explained below.

3.3.1. Definition of inclusion and exclusion criteria

The approaches selected must be related to model consistency management. Approaches which were not based on UML models were excluded. Initially, the selection criteria were interpreted liberally and clear exclusions were only made with regard to title, abstract and introduction.

3.3.2. Procedures for studies selection

By following the indications mentioned in [7], we established a multistage process made up of three stages with different selection criteria:

- In the first stage, the search string must be run on the selected sources. An initial set of studies was obtained from the reading of the title, abstract and introduction of all the studies selected according to the inclusion and exclusion criteria. Studies which were not clearly related to any aspect of the research question were not included.
- In a second stage, the exclusion criteria were based on the following practical issues: short papers, non-English papers, non-International Conference papers and non-International Workshop papers.
- In a third stage, the selection process was based on detailed research questions (see Section 3.1).

Furthermore, and in accordance with [7], a list of studies which had been excluded as a result of the more detailed inclusion/exclusion criteria was be created. This list did not include those irrelevant papers that could be clearly excluded in the first stage after having applied the exclusion criteria.

Finally, please note that the procedure execution and all the studies selected were analyzed by the first two authors of this paper and supervised by the third.

3.3.3. Selection execution

The review process must be documented in sufficient detail [7]. The execution of our selection therefore produced various lists of studies which collected the output of each stage of the selection procedure. The information for each stage of the studies selection procedure was collected in the following manner:

- The first stage produced as output a list for each source which contained all the studies that fulfilled this first stage.
- The second stage produced as output a list of studies for each source which contained all the studies that did not fulfil the second stage inclusion criteria of the procedure for studies selection. These lists contained the excluded work together with the reason for their exclusion from the SLR.
- The third stage produced as output a list of studies for each source which contained all the studies that fulfilled the second stage of the procedure for studies selection. A completed extraction form was included for each work (see Section 3.5).

All the generated lists are available in the complete version of this SLR (see [10]).

3.4. Threats to the validity of this SLR

The main threats to validity in this systematic literature review are related to bias in the selection of the studies to be included and, in some cases, possible inaccuracy in data extraction.

We have considered the five digital libraries mentioned in Section 3.2 which, in turn, include other important electronic sources such as DBLP or CiteSeer. With regard to this point, perhaps the major validity issue facing this systematic literature review is whether we have failed to find all the relevant primary studies, although the scope of conferences and journals covered by the review is sufficiently wide for us to have achieved completeness in the field studied. Nevertheless, we are conscious that it is impossible to achieve total completeness. Some relevant papers may exist which have not been included, although the width of the review and our knowledge of this subject have led us to the conclusion that, if they do exist, there are probably not many.

The selection of papers and data extraction has been carried out by following a multistage process as is suggested in [7]. The three authors of the SLR have participated in this process, and all the selected studies have been analyzed by the firsts two authors and supervised by the third throughout the process. The criteria applied in each stage have been detailed in Section 3.3.2. It is important to note that any systematic literature review is limited to reporting the information provided in the primary studies.

Finally, in order to validate the completeness of the SLR, the studies selected in stage two were reviewed by external experts who detected some important papers that had not been included in the SLR. The primary studies in the SLR have therefore been extended with those papers [16,17] provided by experts in this field. These papers will not be taken into account in the statistics shown in either Tables 2 and 3 or in Fig. 1 since they summarize statistics related to the search process.

3.5. Information extraction

Having defined how the studies were to be selected, it was necessary to design data extraction forms to record the information obtained from the primary studies. The extraction form developed was based on the research questions defined in Section 3.1, general information related to the study identification and certain features defined in [14] related to the objective and subjective results of each study. Each paper's extraction form included the following items:

- Source. Where the paper was found (see Section 3.2).
- Study identification.
- Summary of the paper.
- Inclusion and Exclusion Criteria.
- Research Questions:
  - Method, technique or language used in the paper to manage the model consistency.

Table 2
Summary of the studies selected at each stage of the selection procedure.

	IEEE	ACM	Google Scholar	Science direct	Scopus	Total
Total results	10	170	485	24	218	907
Results selected (stage one)	10	12	24	7	37	90
Results selected (stage two)	4	6	12	5	27	54

Table 3
Summary of search engines overlap in the second stage.

	IEEE	ACM	Google Scholar	Science direct	Scopus
IEEE	–	[18]	[19]	0	[18,20]
ACM	[18]	–	[21,22]	0	[20,18,21,23,22]
Google Scholar	[19]	[21,22]	–	[24]	[18,21,22,24]
Science direct	0	0	[24]	–	[18,24,25]
Scopus	[18,20]	[20,18,21,23,22]	[18,21,22,24]	[18,24,25]	–

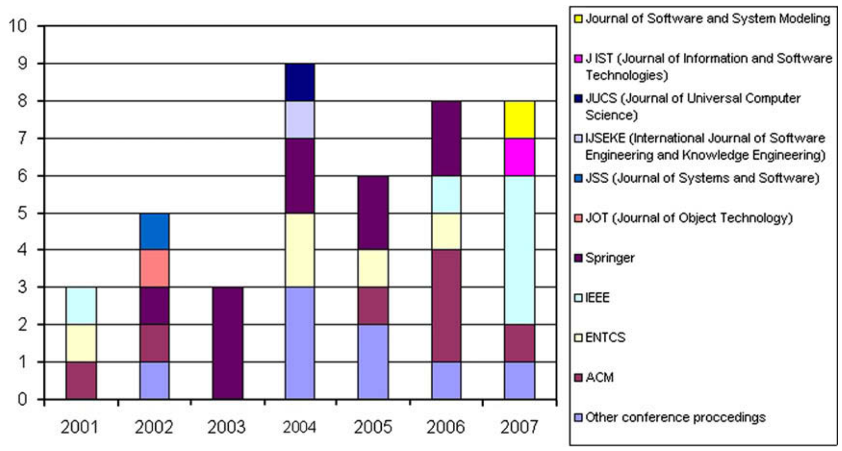

Fig. 1. Number of publications over specific time period.

- UML version. If the approach used a simplified or owned version of the UML metamodel, this field was filled in with "Simple UML".
- Formal approach (Yes/No).
- Diagram support. Types of diagrams supported by the approach.
- Consistency support: Syntactic–Semantic. Horizontal–Vertical.
- Extension mechanism.
- CASE tool integration.
- Automatic support.

– Objective Result Extraction
  - Study Results: effect obtained through the study review.
  - Study Problems: study limitations found by the paper's author.

Table 4
Summary: technique and support.

Reference	Technique	UML version	Formal	Extension mechanism	CASE Integration	Automatic support
Amaya et al. [26]	xLinkit	Simple UML	NO	NO	NO	NO
Amalio et al. [27]	Z	Simple UML	YES	NO	NO	NO
Chiorean et al. [24]	OCL	1.3	NO	OCL	NO	YES, OCLE
Diethers et al. [28]	Timed Automata (UPPAAL model checker)	1.3	YES	NO	YES	YES
Egyed et al. [20,18,23]	Constraints defined with the language/checker available in IBM rational Rose	1.3	NO	NO	YES (IBM Rational Rose)	YES
Egyed et al. [29]	Transformation and comparison	1.3	NO	NO	YES (IBM Rational Rose)	YES
Engels et al. [30,31,25,32–34]	CSP (Communication Sequential Process)	1.3	YES	NO	NO	YES, consistency workbench
Fryz et al. [35]	Graphs	2.0	YES	NO	NO	NO
Graaf et al. [36]	Manual inspection	1.4	NO	NO	NO	NO
Hausmann et al. [37]	Graphs	1.4	YES	NO	NO	NO
van Hee et al. [38]	Petri Nets (PN) and Z	Simple UML	YES	NO	NO	NO
Inverardi et al. [39]	SPIN model checker	Simple UML	YES	NO	NO	YES
Kholkar et al. [40]	SAL (Symbolic Analysis Laboratory)	Simple UML	YES	NO	NO	YES
Laleu et al. [41]	B	Simple UML	YES	NO	NO	YES
Lam et al. [42]	pi-Calculus	2.0	YES	NO	NO	NO
Lucas et al. [43]	Rewriting Logic (Maude)	Simple UML	YES	NO	NO	NO
Malgouyres et al. [22]	CLP (Constraint Logic Programming)	Simple UML	YES	NO	NO	NO
Mens et al. [44]	Graph transformation rules	Simple UML	YES	NO	NO	YES, AGG
Ossami et al. [45]	B	Simple UML	YES	NO	NO	YES, B/Smart Tools
Paige et al. [21,46]	PVS (theorem proving), Eiffel and JML (OO-programming)	Simple UML (BON diagrams)	YES	NO	NO	YES, BON- CASE
Paige et al. [47]	OCL	Simple UML	NO	OCL	NO	NO
Rasch et al. [48]	Object-Z and CSP	1.5	YES	NO	NO	YES, FDR
Sapna et al. [19]	SQL triggers	Simple UML	NO	OCL	NO	NO
Schrefl et al. [17]	Petri Nets	Simple UML	YES	NO	NO	NO
Shinkawa et al. [49]	Colored Petri Nets (CPN)	Simple UML	YES	NO	NO	NO
Spanoudakis et al. [50,51]	OCL	Simple UML	NO	OCL	NO	YES, S-Tool
Straeten et al. [52–55]	Description Logic (DL)	2.0	YES	NO	YES	YES
Wagner et al. [16]	Graphs	1.5	YES	NO	YES, plugin of Fujaba	NO
Wang et al. [56]	FSP (Finite Transition System)	Simple UML	YES	NO	NO	YES
Yao et al. [57]	Petri Net (PN)	Simple UML	YES	NO	NO	YES, Owner tool
Yeung et al. [58]	B and CSP	1.4	YES	NO	NO	NO
Zhao et al. [59]	SPIN model checker	Simple UML	YES	NO	NO	YES

– Subjective Results Extraction
- General Impressions and Abstractions: subjective conclusions after studying the paper.

This form was used for classifying each study. A complete list of the forms filled in for each primary study can be found in [10]. Part of this information is included in the following sections.

3.6. Extraction execution

The systematic literature review took place in March 2008 and no limitations were imposed on the years covered by the search (although the studies which were eventually analyzed were from 2001 to 2007). Table 2 shows a summary of the studies selected in each stage of the selection procedure for each source. The "Total results" are those results which were obtained by running the search string on the selected sources. The next two rows show the results obtained after applying stages one and two of the studies selection procedure. The approaches resulting from this last stage were studied in depth and information concerning the detailed research questions and other fields of the extraction forms was extracted from each paper. Although 56 works were selected, some of them appeared in different sources, so repeated studies were eliminated. 44 works were eventually analyzed (see first column Table 4). Moreover, Fig. 3 summarizes the overlapping search engines produced in the second stage, in order to provide details of how many papers were found in various different sources. Fig. 1 presents an overview of the number of selected publications over a specific period of time, together with their publishers. As the reader can notice, consistency management has attracted the attention of numerous quality journals and conferences in recent years, which denotes that this still is a very active field of research.

4. Results of the systematic literature review

This section shows a summary of the results obtained in the SLR, along with an analysis of the data collected, in order to identify open issues, limitations of the current approaches and recommendations which can be used to offer a suitable model consistency management within the scope of UML.

Once the extraction execution had been completed, it was important to ensure that multiple publications of the same approach were not included in the data analysis. Papers that appeared in multiple sources were thus taken into account only once. Once the duplicated papers had been removed, the different papers describing the same data or approach were grouped together, since duplicate reports would seriously bias the results of the data synthesis [7]. The total number of different approaches included in Tables 4 and 5 is 32.

The results are summarized in two tables. Each of the table's entries groups the papers according to the author's own approach, and these entries have been ordered by the year of publication and the first author's name. Table 4 shows the values regarding to the research questions related to "Technique", "UML version", "Formal", "Extension mechanism", "CASE integration" and "Automatic support". This table therefore shows the general information of each approach, i.e., which method is used and what the tool support offered by each approach is.

Table 5 shows the diagrams supported by each approach and what kind of inconsistency problems they handle. The remaining items included in the extraction form for each approach can be found in [10].

Moreover, Fig. 2 and Table 6 show schematically which diagrams and what consistency support is offered in current literature. Fig. 2 shows the percentage of diagrams involved in the proposals. As we can see, three diagrams (class, state and interac-

Table 5
Summary: diagram and consistency support.

Reference	Diagram support	Consistency support
Amaya et al. [26]	Use cases, class and sequence	Semantic – Horizontal
Amalio et al. [27]	Class and statecharts	Syntactic/Semantic – Horizontal
Chiorean et al. [24]	Class	Syntactic – Horizontal
Diethers et al. [28]	Sequence and statecharts diagrams	Semantic – Horizontal
Egyed et al. [20,18,23]	Class, sequence and statecharts	Syntactic – Horizontal
Egyed et al. [29]	Class	Syntactic – Vertical
Engels et al. [30,31,25,32–34]	State diagrams	Semantic – Horizontal/Vertical
Fryz et al. [35]	Use cases and class	Syntactic – Horizontal
Graaf et al. [36]	Sequence and statecharts	Semantic – Horizontal
Hausmann et al. [37]	Class and objects	Syntactic – Horizontal
Hee et al. [38]	Class, sequence and statecharts	Syntactic – Horizontal
Inverardi et al. [39]	Statecharts and sequence	Semantic – Horizontal
Kholkar et al. [40]	Use cases and class + owner diagrams (see [10] for details)	Semantic – Horizontal
Laleau et al. [41]	Class, statecharts and communication	Syntactic – Horizontal
Lam et al. [42]	Sequence and statecharts	Semantic – Horizontal
Lucas et al. [43]	Class and communication	Syntactic – Horizontal
Malgouyres et al. [22]	Class	Syntactic – Horizontal
Mens et al. [44]	Class and statecharts	Syntactic – Vertical
Ossami et al. [45]	Class and statecharts	Syntactic/Semantic – Horizontal/Vertical
Paige et al. [21,46]	Class and communication	Syntactic/Semantic – Horizontal/Vertical
Paige et al. [47]	Class and sequence	Syntactic – Horizontal
Rasch et al. [48]	Class and statecharts	Semantic – Horizontal
Sapna et al. [19]	Use case, activity, class, sequence and statecharts diagrams	Syntactic – Vertical
Schrefl et al. [17]	Statecharts	Semantic – Horizontal/Vertical
Shinkawa et al. [49]	Use case, sequence, activity and statecharts	Syntactic – Horizontal
Spanoudakis et al. [50,51]	Class and sequence	Syntactic – Horizontal
Straeten et al. [52–55]	Class, sequence and statecharts	Syntactic/Semantic – Horizontal/Vertical
Wagner et al. [16]	Class	Syntactic – Horizontal
Wang et al. [56]	Sequence and statecharts	Semantic – Horizontal
Yao et al. [57]	Sequence and statecharts	Semantic – Horizontal
Yeung et al. [58]	Class and statecharts	Syntactic/Semantic - Horizontal
Zhao et al. [59]	Sequence and statecharts	Semantic – Horizontal

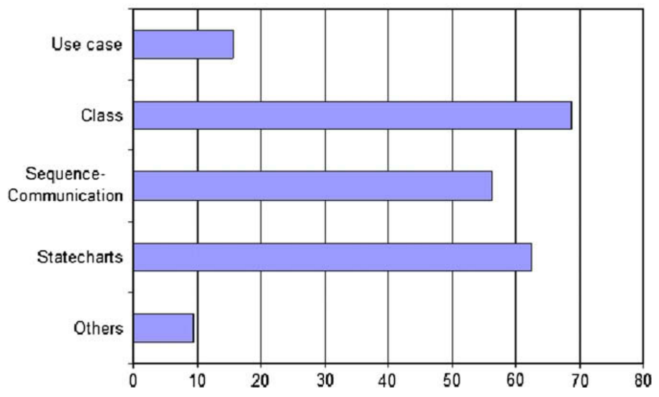

Fig. 2. Percentages of diagram support.

Table 6
Percentages of consistency support.

Type of consistency	Syntactic	Semantic
Horizontal	53.13% (17 of 32)	53,13% (17 of 32)
Vertical	18.75% (6 of 32)	15,63% (5 of 32)

tion diagrams) are tackled by more than 60% of the proposals, whereas the remaining UML diagrams are usually overlooked.

Table 6 shows the percentage of approaches tackling each type of consistency. The table clearly demonstrates that vertical inconsistency problems are studied with less frequency than those which are horizontal.

4.1. Analysis of results, conclusions and recommendations

Having shown the results of the systematic literature review, in this section we provide: (1) our conclusions, which were obtained by analyzing the collected data for each research question and our own experience; and (2) recommendations for possible future research based on the research questions used in this review.

4.1.1. UML version

The majority of the approaches presented (60%) use simplified versions of the UML metamodel. This is a disadvantage with regard to the use of the approaches in industrial software development, but this problem is understandable since many authors do not have the support of companies and their proposals are of an academic nature and solely attempt to fill a gap that exists in current UML modelling. The other proposals (40%) use the UML version available at the time of their publication or the version offered by the tool used for building the models.

Although this is an understandable limitation, it is one that should be rectified in future approaches. A means of solving this might be through the integration of the proposal into a standard software development platform such as EMF (Eclipse Modelling Framework) [60], which allows different metamodels (UML 2.0, simple metamodels, DSLs (Domain Specific Languages), ...) to be worked on since it includes an implementation of EMOF [61].

4.1.2. Formal approach

75% of the techniques used in the proposals for detecting and handling inconsistency problems are formal. None of the formal techniques used stand out above the others, although almost all the proposals that tackle semantic inconsistency problems use a model checker to simulate the behaviour of the system and to find inconsistencies.

This high percentage reveals that the use of formal techniques offers advantages in dealing with consistency problems. Formal techniques add precision to UML models, and their mathematical underpinning permits the use of a wide range of applications such as model checkers, theorem provers, coherence checkers, etc., to name but a few. Many approaches therefore use these methods, in spite of the fact that formal techniques are not yet very popular in the industrial software development community [62]. This unpopularity is usually due to the fact that these approaches are difficult for the modellers to use directly, and that the feedback they offer is usually poor and difficult for non-experts to understand. However, this problem can be solved if the approach is integrated into standard software development platforms [62].

Table 7
Specification technique paradigm.

Reference	Formal technique	Paradigm
Amalio et al. [27]	Z	State transitions
Diethers et al. [28]	Timed Automata (UPPAAL model checker)	State transitions
Engels et al. [30,31,25,32–34]	CSP (Communication Sequential Process)	Process algebra
Hausmann et al. [37]	Graphs	State transitions
Fryz et al. [35]	Graphs	State transitions
Hee et al. [38]	Petri Nets (PN) and Z	State transitions
Inverardi et al. [39]	SPIN model checker	Logic
Kholkar et al. [40]	SAL (Symbolic Analysis Laboratory)	State transitions
Laleu et al. [41]	B	State transitions
Lam et al. [42]	pi-Calculus	Process algebra
Lucas et al. [43]	Rewriting Logic (Maude)	State transitions
Malgouyres et al. [22]	CLP (Constraint Logic Programming)	Logic
Mens et al. [44]	Graph transformation rules	State transitions
Ossami et al. [45]	B	State transitions
Paige et al. [21,46]	PVS (theorem proving)	Logic
Rasch et al. [48]	Object-Z and CSP	State transitions and Process algebra
Schrefl et al. [17]	Petri Nets	State transitions
Shinkawa et al. [49]	Colored Petri Nets (CPN)	State transitions
Straeten et al. [52–55]	Description Logic (DL)	Logic
Wagner et al. [16]	Graphs	State transitions
Wang et al. [56]	FSP (Finite Transition System)	State transitions
Yao et al. [57]	Petri Net (PN)	State transitions
Yeung et al. [58]	B and CSP	State transitions and Process algebra
Zhao et al. [59]	SPIN model checker	Logic

Due to the advantages offered by these methods, we recommend that inconsistency problems be tackled with the use of formal techniques, although a suitable support within a CASE tool should be developed if they are to be used in industrial software development.

Finally, in order to show what the most frequently used formal techniques are, we have classified each technique according to the four paradigms identified in [63]:

- State transitions: the specification describes a transition relation on a set of *states*, e.g. B, Z, Petri Nets, etc.
- Algebra: the specification describes a set of operations defined on a set of types (also called *sorts*). Events are represented by a function (also called an *operation*). The behaviour of functions is given by a set of equations (axioms) which states how functions are related, e.g. CASL.
- Process algebra: this is a special type of algebra. Its operations are applied to elementary processes and events to describe how events may occur, e.g. (e-)LOTOS.
- Logic: the behaviour of functions is given by a set of equations (axioms) which states how functions are related, e.g. PVS, pi-calculus, etc.

More detailed information about these definitions can be found in [63].

In Table 7, each approach has been classified according to this classification. Fig. 3 shows a summary of this information. As the table shows, most of the approaches presented (71%) use a state transitions technique.

4.1.3. Diagram support

The diagrams tackled by each approach alter according to the kind of consistency problem that they wish to check or handle. For example, approaches that deal with behaviour consistency problems usually consider sequence and state diagrams, whereas approaches for static consistency problems usually tackle class diagrams together with other diagrams that use the information defined in the class diagram such as sequence, communication or state diagrams.

As Fig. 2 shows, in general, the interaction (sequence or communication), statecharts and class diagrams are those most frequently tackled, whereas other UML 2.0 diagrams such as timing, activity or components diagrams are not studied. Although this may be a limitation, studies such as [64] demonstrate that these diagrams are those which are most frequently used in industrial software development. The diagrams checked are therefore sufficiently representative to prove the validity of a proposal.

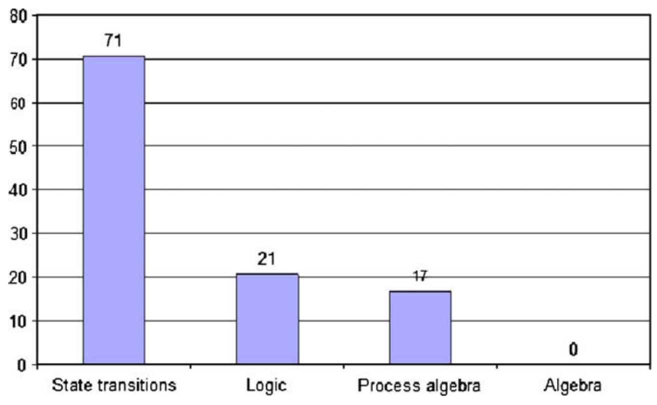

Fig. 3. Summary of formal paradigm.

4.1.4. Consistency support

Table 6 shows a summary of the consistency support offered by the selected proposals. Note that the consistency problems most frequently tackled are syntactic and horizontal problems.

One of the most important conclusions that may be reached by studying this figure is that only a small number of proposals cover problems related to vertical consistency. This kind of problems is particularly important within the scope of MDE approaches in which consistency problems at different levels of abstraction arise. Future proposals dealing with the solution of consistency problems should, therefore, consider this type of inconsistencies.

4.1.5. Extension mechanism

As was previously mentioned, the extensibility of a proposal is related to its usability and its maintainability. However, Table 4 shows that only 12.5% of the approaches offer a means of extending the proposals without being forced to program in the technique used. These proposals use OCL (Object Constraint Language) [65] as a language through which to express consistency problems. However, OCL is too limited to express them. For example, it does not allow us to fix any inconsistencies found since its constraints are side-effect free and cannot modify model elements. Other problems related to OCL, such as the fact that it is not easy to understand, are identified in [66].

On the other hand, as was mentioned in the previous section, the remaining approaches (87.5%) are limited to the implementation offered by the authors, since it is impossible to include all the possible inconsistency checks made by the authors of the approach. This limitation makes expert support necessary if support for new UML diagrams or consistency checks and handling are to be included. This limitation becomes more meaningful within the scope of DSL development, which is rising due to its use in the majority of the proposals connected with MDE. New kinds of models are defined within this scope, so the lack of maintainability also appears when we wish to tackle inconsistency problems within these new models.

This limitation signifies that the solutions proposed for these problems are only partial. However, they could be overcome if the approach were to offer a way in which to extend them through a well-known language, which has sufficient power to express both the definition and the handling of consistency rules or constraints and expressiveness, thus making them easy to read and understand.

4.1.6. CASE tool integration

The studied approaches are usually supported by a tool (53.1%), but only 15.6% of them are suitably integrated within a CASE tool which permits the easy use of the checks implemented. Moreover, the non-integrated approaches in a CASE tool offer poor feedback which is difficult for modellers to understand, since the output of the consistency checks is usually expressed within the scope of the technique chosen (for example, the formal technique used).

Future new approaches should not only facilitate the use of the checks already implemented, but also integrate the definition of new consistency checks and handlings through to the extension mechanism and improve the feedback that the consistency check produces with the aim of easing the modellers task of identifying and handling the problems detected in the models. To do this it will be necessary to select and extend one of the existing CASE tools. The extension capacities of CASE tools should, therefore, be taken into account during the selection process.

4.2. Other future research trends

In addition to the observations concerning each research question which may serve to guide future research in model consistency management, this SLR has also brought to light other considerations, recommendations and open issues which should be taken into account.

4.2.1. Inconsistencies in MDE

Model transformations have always been an important field of research through which to automate (totally or partially) model evolution throughout downstream development, or simply to discover models which improve certain features of the source models (based on certain criteria: experience, metrics...). Furthermore, in recent years, the profound impact of the MDE proposal has meant that model transformation has become a highly active direction for research and development. Since MDE proposes the construction of a system's models at different levels of abstraction, from models that do not contain details of a specific platform to models that take into account the features of the specific platform in which the system will be implemented, inconsistencies in the more abstract models could make (semi-)automatic generation impossible [6]. Moreover, changes in less abstract models may produce inconsistencies with regard to their more abstract models. Vertical consistency therefore becomes one of the most relevant unresolved problem since, as we have already mentioned, it is usually overlooked in the approaches used to handle inconsistency problems.

4.2.2. Handling inconsistencies

A concrete inconsistency problem can be handled by several solutions. Moreover, in most situations, the modeller should decide which solution is the most suitable for the system that is being developed. For instance, if a method used in a sequence diagram does not appear in its corresponding class diagram it is possible to, for example, add a new method to the class of the object that receives the message or modify the method of the message with one from that class. Although, some approaches handle the inconsistencies found, this handling might not be the most suitable for the system, so new approaches should offer the possibility of defining and choosing among multiple possible actions.

5. Our proposal: model consistency management powered by transformation languages

Having taken into consideration the recommendations for surveys provided in [67], in this section we define an approach which attempts to show the viability of overcoming the limitations and unresolved situations identified by the systematic literature review. The features of this approach are focused on offering an approach that will be: (1) extensible, (2) aligned with the MDA proposal, and (3) suitably CASE tool integrated. By taking these aims as our starting point, this section proposes an initial approach based on three elements:

- Transformation languages, which will be the basis for the definition of the extension mechanism and the front-end of the proposal.
- Rewriting logic [8]. Owing to the benefits of formal techniques already commented on, the approach will be implemented over a formal language in order to use its mathematical underpinning to enable theoretical properties to be proved. The formal language used to give support to the whole proposal is Maude [68]. This technique will be the back end of the proposal.
- A CASE tool which appropriately integrates all these technologies based on Eclipse EMF [60].

Since the approach presented is not the main aim of this paper and will be presented in further work, these features will be explained only briefly in the following sections.

5.1. A transformation language for model consistency management

One of the most important limitations of the current approaches used outside academic scopes is the absence of an extension mechanism. This problem can be solved by offering a well-known intermediate representation or language to extend the approach.

Since a consistency problem can be seen as a set of relationships that must be held among model elements conforming to one or several metamodels, and since transformation languages define relationships among the metamodel elements that will be transformed, our proposal consists of reinterpreting the semantics of one of these transformation languages and using it as a basis to define an extensible approach for model consistency management.

Several transformation languages with which to define transformations have appeared within the scope of MDE [69–71]. These definitions are usually expressed as a set of transformation rules that describe how a source model is transformed into a target model [72]. That is, these rules describe how metamodel elements of the source model are transformed into other metamodel elements in the target model. From our point of view, the semantics of these transformation rules will be reinterpreted in order to express relationships that must be held among metamodels, i.e., consistency relationships that must exist among metamodels. An inconsistency problem will thus be expressed as a set of relationships (by means of transformation rules). With these semantics, it would appear to be suitable to use a transformation language such as an intermediate language which is well-known to modellers.

The language chosen for our approach is that of QVT Relations [69], proposed by the OMG (Object Management Group) within the scope of MDA. In this language, relations among metamodels establish how the transformations are carried out. The main reasons for choosing this language as the basis of our approach have been:

- It is one of the languages defined by the OMG in QVT. This guarantees a wide acceptance within the software development community.
- *QVT Relations* is the most abstract and user-friendly language of all the languages defined within the QVT standard [69].
- It is capable of expressing any kind of transformation, and thus any kind of consistency problem among metamodels.

Therefore, QVT Relations provides a well-known and sufficiently expressive language which, when interpreted in this way, provides a mechanism through which to define consistency relationships in industrial software development. Other features of this language are declarative specification and complex object pattern matching, features which are easily supported by our approach.

5.2. The formal language maude

The formal language chosen to specify our approach is Maude [68]. This language is based on equational and rewriting logic and its specifications are executable, thus allowing us to build prototypes, check constraints over a system, and prove theoretical properties over the behaviour of a system in an efficient manner (from half a million to several million rewrites per second [68]).

In rewriting logic [8], a system is specified through a rewrite theory, which consists of a signature Σ (sorts and operations), a set E of equations, and a set of rewriting rules. The static part of a system is modelled by means of equational logic (Σ and E), and the dynamic part is specified by adding rewriting rules, that is, a set of rules which specify how the system's state changes.

One of the most important concepts of rewriting logic that will be used in our approach is the concept of the rewriting rule. A rewriting rule (named l) describes a local concurrent transition that can take place in a system. If the pattern on the left-hand side of the rule (t) matches a fragment of the system state, the matched fragment is transformed into the corresponding state of the right-hand side of the rule (t'), which is expressed as: $l : t \rightarrow t'$.

Maude allows both the specification of this kind of logic and its execution in an object oriented manner, in which the system elements are represented as classes that can be instantiated as objects that exchange messages between each other. Maude also offers several extension modules which can be used in the context of model consistency management. Two of these are the Maude strategies language, which allows us to establish the execution order of the consistency rules, and the Maude metaprogramming capacities, which help us to link rewriting logic with the front end of the proposal.

5.3. Model consistency management through lQVT-Maude language

Fig. 4 summarizes the main elements of which the approach is made up. This figure shows OMG standard elements, the Maude elements, the relationships among them and how an inconsistency problem in natural language will be expressed, firstly as QVT Relations and then as rewriting rules, in the formal back end. The role of each element in this approach will be shown in the following sections by means of an example.

5.3.1. Metamodels specification

The metamodels used in this paper will be specified according to the concepts given in Section 5.2. Metamodel elements will be specified in Maude by means of classes with attributes that describe the metamodels, and their instances (*Maude objects*) represent models that conform to their corresponding metamodel. These relationships are shown in Fig. 4 by means of solid lines between models and metamodels.

In this Figure, two metamodels are involved in the consistency definition. However, in general, more than two metamodels can appear, and the definition may even involve only one metamodel. As an example, in this work we will use metamodels of a simple UML class diagram and a simple UML sequence diagram. Both will be specified in Maude, and their models will be instantiated through *Maude objects*. These metamodels are shown in Fig. 5.

5.3.2. QVT Relations features in maude

In this subsection, we briefly analyze the basis of QVT Relations and how they are also presented in Maude. This concept is summarized in Fig. 6.

QVT Relations is a declarative model transformation language, which means that its implementation using a declarative language such as Maude is more "natural" than other non-declarative languages.

On the one hand, a relation declares constraints that must be satisfied by the metamodels (or domains) that participate in the relation. Each domain establishes a pattern (with a set of variables and constraints) that must be matched with the candidate models in order for the transformation to be carried out. These are known as *object template expressions*, which are directly expressed in Maude since pattern-matching is one of its features.

These QVT Relations will be specified as Maude rewriting rules (see Fig. 4) which change, create or, in this case, verify the elements in the models. Moreover, the constraints express conditions over the models which will be specified as conditions in the rewriting rules.

QVT Relations provide all these elements textually, but once they have been specified in Maude, they are transformed into mathematical entities, thus enabling us to take advantage of the total power of mathematical inference mechanisms, without losing the intuition of the QVT concepts.

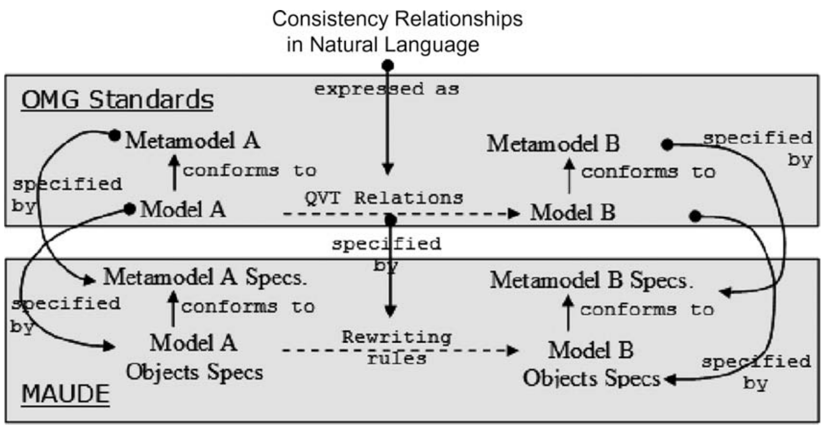

Fig. 4. Summary of the approach elements.

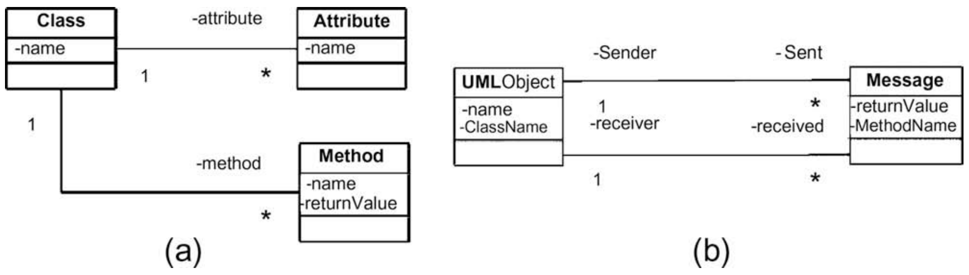

Fig. 5. Simple class diagram (a) and simple sequence diagram (b) metamodels.

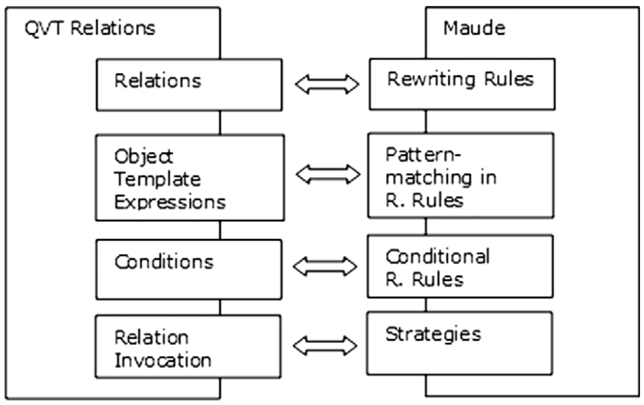

Fig. 6. Equivalence of concepts–features.

```
relation SequenceObjectMessageToClassMethod{
 cn, mn, rv:String;
 checkonly domain sequence o:UMLObject{
  className = cn,
  received = ms:Message {
   methodName = mn, returnValue = rv
  }
 };
 enforce domain class c:Class{
  mame = cn,
  method = mt:Method { name = mn, returnValue = rv }
 };
}
```

Fig. 7. Consistency relation in the use of methods through a QVT relation.

5.3.3. lQVT-Maude language

The main features of QVT Relations can be supported directly in Maude. However, the pattern-matching of Maude is based on the syntax which defines the terms of the model elements, so the means of expressing relations between objects and matching them during their execution is slightly different to that offered by QVT Relations, which would not allow Maude to be directly usable by modellers. In order to solve this problem, we have used Maude's metaprogramming capacities to extend it with a new pattern-matching language called the *lQVT-Maude* (*like-QVT-Maude*) language, which will be used to join the front end of the proposal with the back end.

In order to implement this pattern language based on QVT Relations, Maude also allows us to define the grammar of domain-specific languages (in our case, a simplified version of the QVT Relations pattern-matching grammar, named *lQVT-Maude*). Since the pattern matching and its execution in Maude are based on the Maude grammar, this specific language is not directly executable in Maude. To solve this problem, once the grammar has been defined, Maude allows the specification of *metalanguage* applications, in which Maude parses and handles the new language using its metaprogramming capacities. In our approach we will write rewriting rules, using the *lQVT-Maude* language to define the patterns on the left-hand and right-hand sides of the rules, and metaprogramming is used to handle each rule and to transform it into an executable patterns based on the Maude grammar.

As the following sections will show, rewriting rules and a like-QVT pattern-matching will be used as an extension mechanism for the approach for managing consistency constraints.

5.3.4. Example: consistency between class and sequence diagrams

This section shows (1) how the QVT Relations language can be used to define and handle consistency relationships, and (2) how *lQVT-Maude* is used to express these QVT relations in Maude. To do this, we define an example of a consistency relationship in the use of methods between Sequence and Class Diagrams. In this particular case, a consistent use of a method in the sequence diagram ("methodName") implies that a method exists with the same name in the class of the object ("className") which receives the message, and that the type of the return value is also the same. Fig. 7 shows the QVT Relation that expresses this consistency relationship.

Two domains are involved in this relation: a sequence model and a class model. The rule establishes the relation that must exist among the elements of these models. To do this, it matches objects in each domain that hold the relation. The variables "cn", "mn" and "rv" are used to match the objects of the models. Fig. 8 shows a schema of the objects that hold the relation. If it is not possible to find a set of objects that match the relation, an inconsistency problem exists.

Once the relation has been defined, it will be expressed in Maude through *lQVT-Maude* and rewriting rules. The purpose of the rewriting rules will be firstly to check the consistency relationship and, if this fails, to handle the inconsistency problem. As will be observed, the rewriting rules using *lQVT-Maude* will still be syntactically different from the relation in Fig. 7, since certain implementation details must be considered in the current specification of *lQVT-Maude*. Moreover, the concept of the rewriting rule does not exist within the scope of QVT, but is close to the transformation

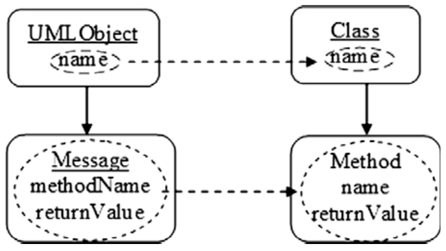

Fig. 8. Schema of the objects matched in the relation.

```
vars cn mn rv : String .
rl [SequenceObjectMessageToClassMethodCheck] :
 domain "sequence" o:UMLObject{
  className = cn,
  *** The "checked" attribute is used to find
  *** messages that have not been checked yet.
  received=ms:Message {
   methodName = mn, returnValue = rv,
   checked = false }
 }
 domain "class" c:Class{ name = cn,
  method=mt:Method { name = mn, returnValue = rv }
 }
=>
 domain "sequence" o:UMLObject{
  className = cn,
  *** If a message holds the consistency relation,
  *** it is marked as checked.
  received=ms:Message {
   methodName = mn, returnValue = rv,
   checked = true }
 }
 domain "class" c:Class{ name = cn,
  method=mt:Method { name = mn, returnValue = rv }
 } .
```

Fig. 9. *lQVT-Maude* rule to check the inconsistency problem.

rule definition, and the use of *lQVT-Maude* brings its syntax closer to that of the QVT Relations.

Fig. 9 shows the *lQVT-Maude* rewriting rule, called "Sequence-ObjectMessageToClassMethodCheck". The left-hand side of this rule matches the objects that fulfil the consistency relation and, once they have been found, the right-hand side of the rule marks the element that must be consistent (the message in this example) as being correct. The extra attribute "checked" (which does not appear in the QVT relation) is used to mark the elements, and sequence diagrams that do not hold the relation will therefore contain unmarked messages after checking has taken place. This is one of the implementation details that must be taken into account in the current implementation of *lQVT-Maude*.

Other implementation details that make the rewriting rule in Fig. 9 different from the relation in Fig. 7 are the definition of variables by following the Maude grammar, certain syntactical differences in the domain definitions or the need to include the left-hand and right-hand sides in the rewriting rule. These differences signify that, future versions of *lQVT-Maude* will be extended to offer a syntax which is closer to the QVT Relations language.

Once the inconsistency has been found, it is necessary to define how to handle it. However, as was mentioned in Section 4.1, a concrete inconsistency may have several solutions. *lQVT-Maude* and rewriting rules are therefore used to allow modellers to define new ways of handling inconsistency problems. In this example, two possible solutions to this inconsistency problem are: to add a new method to the object class that receives the message or to find a candidate method in the class and change it in the message in order to rectify the inconsistency. The specification of only the first solution is shown due to lack of space. Fig. 10 shows the rule which adds the lost element to the source model in order to rectify the inconsistency problem. The left-hand side of the rule shows a pattern that matches the necessary model elements to rectify the inconsistency (in this case, the object and the message in the sequence model, and the class of the object in the class model). The right-hand side of the rule is used to rectify the inconsistency by adding a new element (a method in this example) to the class model. As Fig. 10 shows, the attribute "checked" is used in the left-hand side of the rule to find inconsistent messages previously checked.

Modellers can therefore use lQVT-Maude and rewriting rules to write the rules which are necessary to manage the consistency problems of their models. The operations over these rules (definition, modification, choosing a specific checking and handling ...) will be offered and managed through the CASE tool integration currently being developed (see the following section).

```
vars cn mn rv : String .
*** Once a model has been checked, we can fix the
*** errors found.
rl [SequenceObjectMessageToClassMethod] :
 domain "sequence" o:UMLObject{
  className = cn,
  *** The value of the "checked" attribute is false
  *** for the messages that do not hold the
  *** consistency relation.
  received=ms:Message {
    methodName = mn, returnValue = rv,
    checked = false }
 }
 domain "class" c:Class{
  name = cn, method = mt:Method{}
 }
=>
 domain "sequence" o:UMLObject{
  className = cn,
  *** Once the inconsistency has been fixed, the
  *** value of the "checked" attribute is
  *** established to true.
  received=ms:Message {
    methodName = mn, returnValue = rv,
    checked = true }
 }
 domain "class" c:Class{
  name = cn,
 *** A new method is created in the class named "cn".
  method=mt:Method{ name = mn, returnValue = rv,
    checked = false }
 }
```

Fig. 10. *lQVT-Maude* rule to handle the inconsistency problem.

5.4. A glance at managing semantic inconsistency problems

Finally, this section briefly analyzes how the semantic inconsistency problems can be defined and tackled by only defining transformation rules, in order to show how the presented approach can be also used to manage this kind of problems.

5.4.1. Behaviour definition

Semantic inconsistency problems are related to the behaviour defined by the semantics of the different models (metamodels) involved in the system development. From our point of view, transformation rules (*lQVT-Maude* + rewriting rules) will be used firstly to express how the system behaviour, represented by models, changes. In general, this definition may involve any number of metamodels. For example, if we define an example of a consistency relationship between statemachine and sequence models, we can define their behaviour based on a simulation of the statemachine of each object that appears in the sequence diagram using the received messages as its lifeline. Each message received by an object will thus provoke a change in the state of the statemachine.

5.4.2. Consistency checking

Once the behaviour has been defined, Maude inference mechanisms and commands are used to execute the transformation rules and check these problems. Note that, since different semantic inconsistency problems can be defined over the same behaviour defined, the same behaviour definition can be used to manage all of them. In this example, we can define many possible semantic inconsistency problems between statemachine and sequence models. One of them is to check the consistent behaviour between an object that takes part in the interaction and its associated statemachine. That is, if a sequence of messages received by an object is not supported by its statemachine, an inconsistency problem exists. This problem can be checked by means of the Maude rewriting command, which allows us to use the previous behaviour to simulate a statemachine. This simulation would have two possible terminations: (1) the sequence of messages is fully simulated (meaning that the statemachine and the sequence diagrams are consistent); or (2) a message of the sequence can not be simulated in the statemachine since a transition fired with that message does not exist.

5.4.3. Consistency handling

Once the inconsistency has been found, in our approach model transformations are again used to define how to handle it. In this example, once the inconsistency has been found, several possible solutions to this problem exist. In our case, when an inconsistency is found, the transformation rule will add a new state and a new transition to allow the statemachine to tackle the inconsistent message (this process was used in [73] to create partial statemachine models from sequence models). Note that the next messages to appear in the sequence diagrams will become the new inconsistency problems, since new outgoing transitions in the new created state do not exist, so the checking-handling process will have to be repeated for the remaining messages in the interaction to allow the complete sequence of messages to be tackled by the statemachine model.

Finally, note that if the proposed solution is not useful, the modeller can only check the models and fix them by manual handling or by defining her/his own transformation rule for handling it.

5.4.4. Other maude commands for tackling semantic inconsistency problems

Another Maude inference mechanism could be used to check other inconsistency problems. For example, some Maude tools allow us to check deadlock among any set of rules. Since Maude rewriting rules are used to define the models behaviour, we can use these tools to check deadlock free behaviours. We can therefore, for example, define the behaviour of a statemachine (as in Section 5.4.2) and check that the use of several statemachines for one system is consistent in terms of deadlocks and livelocks.

Thanks to the chosen formalism, the proposed approach thus offers the possibility of tackling both syntactic and semantic problems while most of the approaches studied tackle only one kind since their techniques are more sensitive to each kind of problem. The use of this proposal in semantic consistency will be presented in greater depth in future work.

5.5. CASE tool integration: eclipse EMF

In general, a quite common disadvantage of a formal approach is related to the lack of a suitable and transparent integration with the tools used by the modellers during the development life cycle.

We have chosen the EMF [60] for our implementation. This has become one of the most frequently used modelling tools in academic and industrial environments [74,75] due to its plug-ins mechanism. This mechanism makes it easy to reuse and to add functionality. Another of the most important advantages of using EMF is the full and easy access that it offers to its model and metamodel representation, which is essential for tackling the implementation of our proposal.

With regard to our approach, we use the Ecore metamodel of EMF both to support UML 2.x, and to define new metamodels and models. Moreover, we use MOFScript [76] to transform these metamodels and models into code (in this case, Maude specifications from metamodels and objects from specific models). The Eclipse plug-in mechanism is useful in that it offers *lQVT-Maude* language directly in the EMF environment and a suitable integration of all the elements involved in the approach. The main features that this environment offers are: (1) the possibility of selecting the inconsistency checks that will be executed, (2) the possibility of selecting a solution for inconsistency handling once a problem has been detected, (3) the definition of new consistency problems and how to rectify them, and (4) suitable feedback to the modeller with regard to the activities executed. This integration is still under development and will be shown in its entirety in future work.

Table 8
Answers for the research questions of the proposal.

Research question	RQ value	Technique used
UML version	2.x	Eclipse EMF
Formal approach	YES	Maude
Diagram support	UML 2.x, DSLs,...	Eclipse EMF
Consistency support	Syntactic/Semantic – Horizontal/Vertical	lQVT-Maude rewriting rules
Extension mechanisms	YES	QVT Relations
CASE tool integration	YES	Eclipse EMF
Automatic Support	YES	Eclipse

5.6. Research questions for the proposal

This section summarizes how the different research questions defined in the SLR are answered in the preliminary proposal presented (see Table 8). All the features of the proposal and its use for tackling inconsistency problems will be presented in depth in future work.

6. Conclusions and further work

This work presents the results obtained after carrying out a systematic literature review of literature whose aim was to identify and evaluate the current approaches for model consistency management. To do this, a total of 907 papers published in literature and extracted from the most relevant scientific sources were considered, of which 44 were eventually analyzed in depth, in accordance with the SLR process adopted.

The results of this SLR show that UML model consistency is a highly active and promising line of research, in which a great deal of quality work has been done, but in which some important gaps and limitations still exist, which should be tackled in future work. With regard to the new UML model consistency management proposals, we have three main recommendations to make (see Section 4 for more details). From our point of view, these proposals should:

(RQ1) Include a mechanism to extend the proposal in order to facilitate the managing of new models and inconsistency problems.
(RQ2) Tackle inconsistency problems related to vertical consistency, since they have been less frequently studied.
(RQ3) Fully integrate of all the features of the proposal within a CASE tool. This will thus permit its use and validation outside the academic scope.

Following these conclusions, we have presented a preliminary formal approach for the handling of inconsistency problems. This approach is based on transformation languages and rewriting logic and attempts to solve the limitations identified by the SLR while simultaneously including the strengths of the existing proposals. The main features included in this proposal are: extensibility, alignment with the MDA proposal, a formal basis and CASE tool integration.

The approach presented aims to fulfil the research challenges identified in the SLR through the definition of an extension mechanism by using the QVT Relations syntax. This mechanism will allow us to tackle any type of diagram, since the QVT Relations language can be applied over any metamodel (or DSL), in order to add new consistency checks and to define new ways of handling an inconsistency problems.

Furthermore, vertical inconsistency problems, which have not been studied in depth by the current approaches with regard to the SLR results, can be tackled in a natural manner thanks to the alignment of our proposal with MDA and transformation languages, which tackle models at different levels of abstraction. Since model transformation languages are well-known in this scope, the approach presented is particularly useful for modellers who are familiar with model-driven engineering processes.

Moreover, the use of rewriting logic in this approach, which manages all the elements as mathematical entities, offers a powerful manner in which to verify type properties and the correctness of the models or to use the inference mechanisms of this formalism without losing the legibility, practicality, and expressivity of other known languages such as QVT Relations.

As regards future work, we first intend to extend the *lQVT-Maude* language to avoid having to consider the implementation details mentioned. The distance between QVT Relations and the *lQVT-Maude* language will thus be smaller still. Moreover, this language is beginning to be used in more complex examples and in other kinds of inconsistency problems. What is more, since rewriting rules allow us to transform model elements, their use in offering formal automatic support for model transformations within the scope of QVT and their applications to the verification and validation of transformations will also be studied in future work.

References

[1] OMG, Object Management Group. Unified Modeling Language: Superstructure. Version 2.1.1, Retrieved from: <http://www.omg.org/uml>, 2007.
[2] J. Muskens, R. Bril, M. Chaudron, Generalizing consistency checking between software views, in: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), 2005, pp. 169–180.
[3] Z. Huzar, L. Kuzniarz, G. Reggio, J.L. Sourrouille (Eds.), Proceedings of Workshop on Consistency Problems in UML-based Software Development II, 2003.
[4] D. Schmidt, Guest editor's introduction: model-driven engineering, IEEE Comput. 39 (2) (2006) 25–31.
[5] OMG, MDA Guide Version 1.0.1, <http://www.omg.org/mda>, 2001.
[6] J. Simmonds, C.M. Bastarrica, A tool for automatic UML model consistency checking, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 2005.
[7] B. Kitchenham, Guidelines for performing systematic literature reviews in software engineering, EBSE Technical Report EBSE-2007-01, Software Engineering Group, School of Computer Science and Mathematics, Keele University, UK and Department of Computer Science, University of Durham, Durham, UK, 2007.
[8] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification. Equations and Initial Semantic, Springer-Verlag, 1985. ISBN: 3-540-13718-1.
[9] G. Spanoudakis, A. Zisman, Inconsistency management in software engineering: survey and open research issues, in: S.K. Chang (Ed.), Handbook of Software Engineering and Knowledge Engineering, vol. 1, World Scientific Publishing Co., 2001, pp. 329–380.
[10] F.J. Lucas, F. Molina, A. Toval, A systematic review of UML model consistency management, Technical Report LSI 1-2008, Departamento de Informática y Sistemas, University of Murcia, (Also available from: <http://express.inf.um.es/fjlucas/>), December 2008.
[11] Y. Shinkawa, Inter-Model Consistency in UML Based on CPN Formalism, XIII Asia Pacific Software Engineering Conference (APSEC'06), 2006.
[12] Z. Huzar, L. Kuzniarz, G. Reggio, J.L. Sourrouille, Consistency problems in UML-based software development, in: N. Jardim Nunes, B. Selic, A. Silva, A. Toval (Eds.), UML Modeling Languages and Applications, UML 2004 Satellite Activities Lisbon, October 11–15, LNCS, vol. 3297, Springer Verlag, Portugal, 2004. Revised Selected Papers.
[13] G. Engels, J.M. Küster, R. Heckel, L. Groenewegen, A methodology for specifying and analyzing consistency of object-oriented behavioral models, in: Proceedings of Eighth European Software Engineering Conference Held Jointly with Nineth ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE2001), September 2001, ACM Press, Vienna, Austria, 12, 58, pp. 186–195.
[14] J. Biolchini, P. Mian, A. Natali, G. Travassos, Systematic review in software engineering, Technical Report RT-ES679/05, Universidade Federal do Rio de Janeiro, Program de Engenharia de Sistemas e Computačáo, 2005.
[15] M. Gogolla, Benefits and problems of formal methods, in: Reliable Software Technologies – Ada-Europe 2004, Proceedings of the 9th Ada-Europe International Conference on Reliable Software Technologies, Palma de Mallorca, Lecture Notes in Computer Science, Springer, Spain, 2004, pp. 1–15.
[16] R. Wagner, H. Giese, U. Nickel, A plug-in for flexible and incremental consistency management, in: Proceedings of the International Conference on the Unified Modeling Language 2003 (Workshop 7: Consistency Problems in UML-based Software Development), San Francisco, USA, Technical Report, Blekinge Institute of Technology, San Francisco, 2003.
[17] M. Schrefl, M. Stumptner, Behavior-consistent specialization of object life cycles, ACM Trans. Softw. Eng. Methodol. 11 (1) (2002) 92–148.
[18] A. Egyed, Fixing inconsistencies in UML design models, in: 29th International Conference on Software Engineering (ICSE'07), 2007, pp. 292–301.
[19] P.G. Sapna, H. Mohanty, Ensuring consistency in relational repository of UML models, in: 10th International Conference on Information Technology (ICIT 2007), 2007, pp. 217–222.
[20] A. Egyed, UML/Analyzer: a tool for the instant consistency checking of UML models, in: 29th International Conference on Software Engineering (ICSE'07) 00, 2007, pp. 793–796.
[21] R.F. Paige, P.J. Brooke, J.S. Ostroff, Metamodel-based model conformance and multiview consistency checking, ACM Trans. Softw. Eng. Methodol. 16 (3) (2007) 11.
[22] H. Malgouyres, G. Motet, A UML model consistency verification approach based on meta-modeling formalization, in: SAC'06: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM, USA, 2006, pp. 1804–1809.
[23] A. Egyed, Instant consistency checking for the UML, in: ICSE'06: Proceedings of the 28th International Conference on Software Engineering, ACM, New York, NY, USA, 2006, pp. 381–390.
[24] D. Chiorean, M. Pasca, A. Cârcu, C. Botiza, S. Moldovan, Ensuring UML models consistency using the OCL environment, Electr. Notes Theor. Comput. Sci. 102 (2004) 99–110.
[25] J.M. Küster, Towards inconsistency handling of object-oriented behavioral models, Electr. Notes Theor. Comput. Sci. 109 (2004) 57–69.
[26] P. Amaya, C. Gonzalez, J.M. Murillo, Towards a subject-oriented model-driven framework, Electr. Notes Theor. Comput. Sci. 163 (1) (2006) 31–44.
[27] N. Amálio, S. Stepney, F. Polack, Formal proof from UML models, in: Formal Methods and Software Engineering, Proceedings of the 6th International Conference on Formal Engineering Methods, ICFEM 2004, Lecture Notes in Computer Science, Springer, Seattle, WA, USA, 2004, pp. 418–433.
[28] K. Diethers, M. Huhn, Vooduu: verification of object-oriented designs using UPPAAL, in: Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of the 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, 2004, pp. 139–143.
[29] A. Egyed, Consistent adaptation and evolution of class diagrams during refinement, in: Fundamental Approaches to Software Engineering, Proceedings of the 7th International Conference, FASE 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004 Barcelona, LNCS, Springer, Spain, 2004, pp. 37–53.
[30] G. Engels, R. Heckel, J.M. Küster, L. Groenewegen, Consistency-preserving model evolution through transformations, in: UML 2002 – The Unified Modeling Language, 5th International Conference, Dresden, Germany, Lecture Notes in Computer Science, Springer, 2002, pp. 212–226.
[31] G. Engels, J.M. Küster, R. Heckel, L. Groenewegen, Towards consistency-preserving model evolution, in: IWPSE'02: Proceedings of the International Workshop on Principles of Software Evolution, ACM, New York, NY, USA, 2002, pp. 129–132.
[32] J.M. Küster, G. Engels, Consistency management within model-based object-oriented development of components, in: Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4–7, Revised Lectures, Lecture Notes in Computer Science, Springer, 2003, pp. 157–176.
[33] G. Engels, J.M. Küster, R. Heckel, L. Groenewegen, A methodology for specifying and analyzing consistency of object-oriented behavioral models, SIGSOFT Softw. Eng. Notes 26 (5) (2001) 186–195.
[34] R. Heckel, J.M. Küster, Behavioral constraints for visual models, Electr. Notes Theor. Comput. Sci. 50 (3) (2001) 257–265.
[35] L. Fryz, L. Kotulski, Assurance of system consistency during independent creation of UML diagrams, in: International Conference on Dependability of Computer Systems (DepCoS-RELCOMEX 2007), IEEE Computer Society, Szklarska Poreba, Poland, 2007, pp. 51–58.
[36] B. Graaf, A. van Deursen, Model-driven consistency checking of behavioural specifications, in: Proceedings – Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES, 2007.
[37] J.H. Hausmann, R. Heckel, S. Sauer, Extended model relations with graphical consistency conditions, in: Proceedings UML 2002 Workshop on Consistency Problems in UML-based Software Development, Blekinge Institute of Technology, 2002, pp. 61–74.
[38] K.M. van Hee, N. Sidorova, L.J. Somers, M. Voorhoeve, Consistency in model integration, in: Business Process Management: Proceedings of the Second International Conference, BPM 2004, Potsdam, Germany, 2004, pp. 1–16.
[39] P. Inverardi, H. Muccini, P. Pelliccione, Automated check of architectural models consistency using SPIN, in: 16th IEEE International Conference on Automated Software Engineering (ASE 2001), Coronado Island, San Diego, CA, USA, 2001, pp. 346–349.
[40] D. Kholkar, G.M. Krishna, U. Shrotri, R. Venkatesh, Visual specification and analysis of use cases, in: SoftVis'05: Proceedings of the 2005 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2005, pp. 77–85.
[41] R. Laleau, F. Polack, Using formal metamodels to check consistency of functional views in information systems specification, Inf. Softw. Technol. 50 (7-8) (2008) 797–814.
[42] V.S.W. Lam, J.A. Padget, Consistency checking of sequence diagrams and statechart diagrams using the pi-calculus, in: Integrated Formal Methods, Proceedings of the 5th International Conference, IFM 2005, Eindhoven, The Netherlands, 2005, pp. 347–365.
[43] F.J. Lucas, A. Toval, A precise approach for the analysis of the UML models consistency, in: BP-UML'05: 1st International Workshop on Best Practices of UML, in 24th International Conference on Conceptual Modeling (ER 2005), Klagenfurt (Austria), ISBN: 3-540-29395-7. LNCS 3770, Springer.
[44] T. Mens, R.V.D. Straeten, M. D'Hondt, Detecting and resolving model inconsistencies using transformation dependency analysis, in: Model Driven Engineering Languages and Systems, Proceedings of the 9th International Conference, MoDELS 2006, LNCS, Springer, Genova, Italy, 2006, pp. 200–214.
[45] D.D.O. Ossami, J.-P. Jacquot, J. Souquières, Consistency in UML and B Multi-view specifications, in: Integrated Formal Methods, Proceedings of the 5th International Conference, IFM 2005, LNCS, Springer, Eindhoven, The Netherlands, 2005, pp. 386–405.

[46] R.F. Paige, L. Kaminskaya, J.S. Ostroff, J. Lancaric, BON-CASE: an extensible CASE tool for formal specification and reasoning, J. Object Technol. 1 (3) (2002) 77–96.
[47] R.F. Paige, D.S. Kolovos, F. Polack, Refinement via consistency checking in MDA, Electr. Notes Theor. Comput. Sci. 137 (2) (2005) 151–161.
[48] H. Rasch, H. Wehrheim, Checking consistency in UML diagrams: classes and state machines, in: Formal Methods for Open Object-Based Distributed Systems, Lecture Notes in Computer Science, Springer, 2003.
[49] Y. Shinkawa, Inter-model consistency in UML based on CPN formalism, in: XIII Asia Pacific Software Engineering Conference (APSEC'06), 2006.
[50] G. Spanoudakis, K. Kasis, F. Dragazi, Evidential diagnosis of inconsistencies in object-oriented designs, Int. J. Softw. Eng. Knowl. Eng. 14 (2) (2004) 141–178.
[51] G. Spanoudakis, H. Kim, Diagnosis of the significance of inconsistencies in object-oriented designs: a framework and its experimental evaluation, J. Syst. Softw. 64 (2002) 3–22.
[52] R.V.D. Straeten, V. Jonckers, T. Mens, A formal approach to model refactoring and model refinement, Softw. Syst. Model. 6 (2) (2007) 139–162.
[53] R.V.D. Straeten, M. D'Hondt, Model refactorings through rule-based inconsistency resolution, in: SAC'06: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2006, pp. 1210–1217.
[54] R.V.D. Straeten, Inconsistency detection between UML models using racer and nRQL, in: ADL'04 Third International Workshop on Applications of Description Logics, 2004.
[55] R. Van Der Straeten, T. Mens, J. Simmonds, V. Jonckers, Using description logics to maintain consistency between UML models, in: P. Stevens, J. Whittle, G. Booch (Eds.), UML 2003 – The Unified Modeling Language, Lecture Notes in Computer Science, vol. 2863, Springer-Verlag, 2003, pp. 326–340.
[56] H. Wang, T. Feng, J. Zhang, K. Zhang, Consistency check between behaviour models, in: ISCIT 2005 – Wab International Symposium on Communications and Information Technologies 2005, Proceedings II, 2005.
[57] S. Yao, S.M. Shatz, Consistency checking of UML dynamic models based on petri net techniques, in: 15th International Conference on Computing (CIC'06) 0, 2006, pp. 289–297.
[58] W.L. Yeung, Checking consistency between UML class and state models based on CSP and B, J. Universal Comput. Sci. (J. UCS) 10 (11) (2004) 1540–1559.
[59] X. Zhao, Q. Long, Z. Qiu, Model checking dynamic UML consistency, in: Formal Methods and Software Engineering, Proceedings of the 8th International Conference on Formal Engineering Methods, ICFEM 2006, Lecture Notes in Computer Science, Springer, Macao, China, 2006, pp. 440–459.
[60] E.M.F. Project, Eclipse Modeling Framework (EMF), <http://www.eclipse.org/modeling/emf/>, 2007.
[61] OMG, MOF MetaObject facility specification, Object Management Group., Retrieved from: <http://www.omg.org/docs/ptc/04-10-15.pdf>, 2004.
[62] B. Beckert, T. Hoare, R. Hähnle, D.R. Smith, C. Green, S. Ranise, C. Tinelli, T. Ball, S.K. Rajamani, Intelligent systems and formal methods in software engineering, IEEE Intelligent Syst. 21 (6) (2006) 71–81.
[63] H. Habrias, M. Frappier, Software Specification Methods, iSTE, 2006.
[64] B. Dobing, J. Parsons, How UML is used, Commun. ACM 49 (5) (2006) 109–113.
[65] J. Warmer, A. Kleppe, The Object Constraint Language: Precise Modelling with UML, Addison-Wesley, 1999.
[66] J.L. Sourrouille, G. Caplat, Constraint checking in UML modeling, SEKE (2002) 217–224.
[67] A.J. Smith, The task of the referee, IEEE Comput. 23 (4) (1990) 65–71.
[68] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcote, Maude 2.3 Manual., <http://maude.csl.sri.com/>, 2007.
[69] OMG, MOF QVT Final Adopted Specification, Object Management Group, Retrieved from: <http://www.omg.org/docs/ptc/07-07-07.pdf>, 2007.
[70] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, P. Valduriez, Atl: a qvt-like transformation language, 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, Portland, Oregon, USA.
[71] J.S. Cuadrado, J.G. Molina, Building domain-specific languages for model-driven development, IEEE Softw. 24 (5) (2007) 48–55.
[72] A.G. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[73] P. Selonen, K. Koskimies, M. Sakkinen, Transformation between UML diagrams, J. Database Manag. 14 (3) (2003) 37–55.
[74] Borland, Together: Visual Modeling for Software Architecture Design, <http://www.borland.com/together>, 2007.
[75] Rational, Rational Software Architect, <http://www-306.ibm.com/software/awdtools/architect/swarchitect/>, 2008.
[76] Eclipse, Generative Modeling Technologies (GMT): MOFScript, <http://www.eclipse.org/gmt/>, 2007.