Software & Systems Modeling (2019) 18:2313–2360
https://doi.org/10.1007/s10270-018-0675-4

REGULAR PAPER

Execution of UML models: a systematic review of research and practice

Federico Ciccozzi[1] · **Ivano Malavolta**[2] · **Bran Selic**[3]

Received: 11 July 2017 / Revised: 23 March 2018 / Accepted: 31 March 2018 / Published online: 10 April 2018

Abstract
Several research efforts from different areas have focused on the execution of UML models, resulting in a diverse and complex scientific body of knowledge. With this work, we aim at identifying, classifying, and evaluating existing solutions for the execution of UML models. We conducted a systematic review in which we selected 63 research studies and 19 tools among over 5400 entries by applying a systematic search and selection process. We defined a classification framework for characterizing solutions for UML model execution, and we applied it to the 82 selected entries. Finally, we analyzed and discussed the obtained data. From the analyzed data, we drew the following conclusions: (i) There is a growing scientific interest on UML model execution; (ii) solutions providing translational execution clearly outnumber interpretive solutions; (iii) model-level debugging is supported in very few cases; (iv) only a few research studies provide evidence of industrial use, with very limited empirical evaluations; (v) the most common limitation deals with coverage of the UML language. Based on these observations, we discuss potential research challenges and implications for the future of UML model execution. Our results provide a concise overview of states of the art and practice for UML model execution intended for use by both researchers and practitioners.

Keywords UML · Model execution · Code generation · Model compilation · Model interpretation · Systematic review

1 Introduction

Standardized by the Object Management Group (OMG) in 1997, the Unified Modeling Language (UML)[1] has emerged and established itself as both a de facto and a de jure standard in industrial development of software systems [33]. This was due in part to its versatility, which enables its use as general-purpose language, and also to its ability to be customized through its profiling mechanisms [1] to directly support concepts specific to individual domains, organizations, or projects.

There is evidence that, in industrial practice, UML models have been used primarily for problem understanding (i.e., analysis) and documentation [24], despite the fact that a number of tools support executable variants of UML. These relied on custom semantics in combination with traditional third-generation programming languages, such as C++ or Java, for specifying detailed action code. Consequently, they were not fully compliant with the UML standard, forcing users into a potentially precarious "vendor lock-in" predicament. However, the introduction of UML2 together with the definition of (i) a formal specification of the executable semantics for a subset of UML2, via the Foundational Subset For Executable UML Models (fUML)[2] and (ii) a textual action language, the Action Language for Foundational UML (Alf)[3], enabled compact and complete specification of complex behaviors including their algorithmic parts, such that the models based on these are fully executable provided that corresponding tools are available.

Communicated by Dr. Jeff Gray.

✉ Federico Ciccozzi
federico.ciccozzi@mdh.se

Ivano Malavolta
i.malavolta@vu.nl

Bran Selic
selic@acm.org

[1] School of Innovation, Design and Engineering (IDT), Mälardalen University, Västerås, Sweden

[2] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

[3] Malina Software Corporation, Ottawa, Canada

[1] http://www.uml.org.

[2] http://www.omg.org/spec/FUML.

[3] http://www.omg.org/spec/ALF.

As of today, a plethora of methods and techniques for executing UML models have been proposed (e.g., [26,31,47]). Each of these uses at least one of two possible execution strategies: *translational* or *interpretive* (see Sect. 2). However, they do this in idiosyncratic ways, so that each is applicable only in specific contexts, focusing on specific concerns and problem areas. Consequently, it is currently difficult to have a clear view of existing solutions for the execution of UML models and their characteristics. This makes it difficult for practitioners to decide whether and how to take advantage of executable UML models.

The **goal** of this study is to identify, classify, and evaluate the states of the art and practice of solutions for executing models based on the UML family of languages (i.e., UML2, UML2 profiles, fUML, Alf). With this objective in mind, we performed a systematic investigation of trends, technical characteristics, available evidence, and limitations of current solutions for the execution of UML models in both research and industrial practice.

The main **contributions** of this study are:

- a *classification framework* for classifying, comparing, and evaluating solutions for executing UML models;
- a *systematic review* of the states of the art and practice in executing UML models;
- an exploration of *emerging research challenges and implications* for future research and practice of executing UML models.

The intended **audience** of this study includes both (i) researchers willing to further contribute to this research area (e.g., by defining new solutions for executing UML models or improving existing ones) and (ii) practitioners who want to better understand existing and future solutions for executing UML models in order to select the one that best suits their needs.

From a **methodological perspective**, in this study we perform a *systematic review* [27,51]. As part of our systematic process, we first selected 82 primary studies from over 5400 candidates for answering the research questions that we identified (see Sect. 3.1). Next, we defined a classification framework for characterizing solutions for executing UML models, which we applied to the 82 selected studies. Finally, we analyzed the obtained data to produce: (i) an overview of the states of the art and practice, and (ii) a list of emerging and future research challenges and implications. We expect that the results of these activities will (i) provide a concise overview of the states of the art and practice of UML model execution, and (ii) help researchers and practitioners in identifying the limitations of and gaps in current research as well as the potential and applicability of UML model execution in practice [28].

It is worth noting that, besides numerous systematic reviews and surveys on UML-based methods and techniques exist, we could not identify any systematic study on the execution of UML models. The only work dealing with a comparison of approaches for execution of UML, is represented by Gotti and Mbarki [20], where the authors only compare four approaches selected ad hoc. Consequently, to the best of our knowledge, this study represents the *first systematic investigation of the states of the art and practice of the execution of UML models*.

Outline In Sect. 2 we review the requisite background that provides the context of our study. Section 3 describes in detail the choices we made in designing, conducting, and documenting the study. The results of the study are discussed in Sects. 4, 5, 6, 7, and 8. Section 9 focuses on the main findings and implications for future researchers and practitioners working on UML model execution. Section 10 provides a brief summary of the work and conclusions reached.

2 Background

2.1 From programming to MDE

In the early 1950s, executables were written directly in machine code language. The need for simplification by abstracting out excessive detail about the underlying hardware led to the creation of numerous programming languages, many of which are still in use. These languages used higher-level abstractions to capture desired behaviors. Thus, programming progressed from first-generation machine code languages, to second-generation assembler languages, to what are known as third-generation or high-level programming languages (3GLs). Programs written using second- and third-generation programming languages were not directly executable, but needed to be transformed into a format (i.e., machine code) that the machine could execute. This could be done in one of two ways:

- *Interpretation* the software program is interpreted by a language-specific virtual machine program executing on the target platform;
- *Translation* the software program is translated into *object code*[4] that could be executed directly on the target machine.

However, increasing demands for ever more sophisticated software functionality has led to calls for more powerful

[4] Object code is what a compiler produces. We use this term since it includes both *machine code languages* (e.g., Java bytecode) and *intermediate languages* (e.g., LLVM intermediate representation).

and more flexible methods of programming computers. As before, this meant moving even further away from the concepts of the underlying computing machinery toward methods of expression that are closer to the problem domain—a trend that is often referred to as "raising the level of abstraction" of programming.

The various developments grouped under the common heading of *Model-Driven Engineering* (MDE)[5] were conceived with this goal in mind. At the core of MDE is the notion that models should serve as primary artifacts in software development, as opposed to being merely optional support facilities for analysis and documentation. Abstracting away unneeded (often target-related) details, models can tame complexity and allow developers to better focus on the key design concerns of their applications [44]. In other words, models allow developers (i.e., not necessarily experts in programming) to define complex functions in a more human-centric way than if using traditional programming languages.

Models conform to a metamodel, defined as part of a modeling language which can be either general purpose or domain specific. From the myriad of general-purpose and domain-specific modeling languages (DSMLs) that have been defined to date, UML has emerged and established itself for industrial modeling [33]. UML is general purpose, but it provides powerful profiling mechanisms to constrain and extend the language to achieve DSMLs, via UML profiles. In this paper, we focus on execution of UML and UML profiles. If computer-based models of software systems are specified using an executable modeling language, it is possible to gradually evolve them starting with very abstract models, which may be suitable for "descriptive" purposes, into prescriptive ones that are precise and detailed enough to be executable. However, supporting the full development cycle with a single modeling language presents a very unique language design challenge: the ability to create both highly abstract (and potentially incomplete) descriptive models and very detailed prescriptive ones.

One of the main goals of the MDE paradigm is to promote automation during the development process through computer-based model manipulation of models and related artifacts. A prominent example of this is the automatic generation of executable artifacts from design models by means of model transformations; there is documented evidence that the automatic production of executables from models is a key feature in MDE [25].

By shifting the focus of the development from hand-written code to models, one might reasonably expect that corresponding execution mechanisms, namely *interpretation* and *translation (to object code)*, would be provided for modeling languages as well. However, for a number of reasons, including the need to exploit available middleware, expertise, and development tools, translation of models has primarily targeted high-level languages (e.g., 3GLs) instead of object code (see Fig. 1). Translation to 3GLs can be found also in execution of 3GLs;[6] an example is Cfront, which translated C++ to C to enable the use of C compilers.

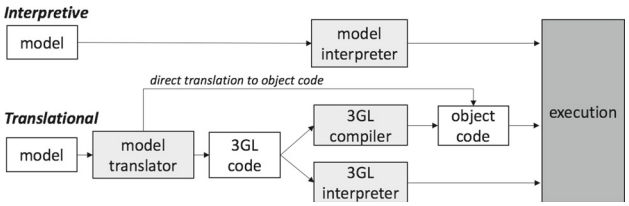


Fig. 1 Model execution strategies

2.2 UML as an executable modeling language

Although the initial release of UML helped in raising the level of abstraction, it was neither sufficiently powerful nor sufficiently precise to produce complete executable programs.

Origins of UMLUML was introduced in the second half of the 1990s as an evolution of object-oriented programming. Its origins are at the intersection of three methodologies popular in the 1980s and early 1990s: the Booch method, Rumbaugh's Object-Modeling Technique (OMT), and Jacobson's Object-Oriented Software Engineering (OOSE). UML 1.x was created and standardized by the OMG in 1997. With UML 1.5, an action semantics for UML was introduced. Nevertheless, until version 2.0, UML was still considered ambiguous and partially inconsistent, especially in terms of execution semantics.

UML 2.0 and above Version 2.0, together with the standardisation of (i) the Foundational Subset For Executable UML Models (fUML), which gives a precise execution semantics to a subset of UML limited to composite structures, classes and activities (inclusion of UML state machines is under formalization too) [48], and (ii) a textual action language, Alf, to express complex execution behaviors, has made UML a full-fledged implementation-capable language [45].

fUML and AlffUML is a standard that defines the execution semantics of UML (e.g., how control structures conditionally execute statements); application models designed with fUML are executable by definition. fUML is intended for the definition of the execution semantics for any DSML based on UML profiles [48]. The work from Mayerhofer et al. [35] has made it possible to use fUML for defining execution semantics of any DSML based on the Meta-Object Facility standard

[5] Other terms used for this purpose include Model-Based (Software) Engineering and Model-Driven Development.

[6] In programming, this is called source-to-source compilation or transpilation.

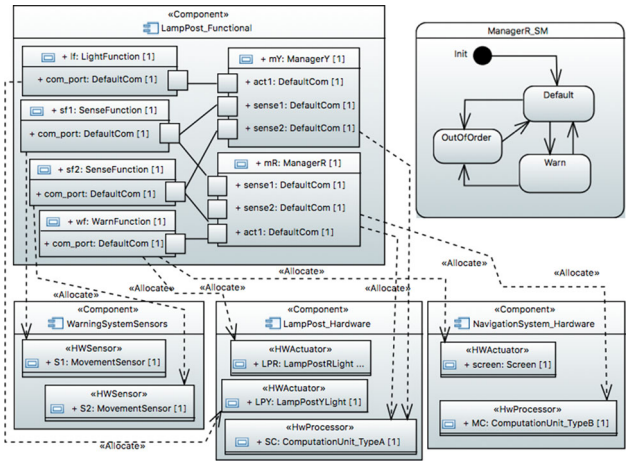


Fig. 2 Executable UML model (from [10])

too. Alf is a textual surface representation for UML modeling elements, whose execution semantics is given by mapping Alf's concrete syntax to the abstract syntax of fUML. While Alf maps to fUML in order to provide its execution semantics, its use is not limited to the context of models conforming to the fUML subset. For example, using (f)UML and Alf, it is possible to fully describe a software functionally, while exploiting the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [34] for modeling hardware components as well as allocations of software to hardware.

Example of executable UML model In Fig. 2 we can see a portion of a UML model representing the smart street lightning system in a concrete graphical syntax, from which executable C++ code can be automatically generated [10].
More specifically, the portion represents a lamppost system in terms of its software functionalities, physical devices and allocations. In terms of UML, `LampPost_ Functional` represents the root software composite component, which contains six software components. Connections between software functionalities are achieved through connectors via ports. Behavioral descriptions of the software components are defined in terms of UML state machines, for defining the overall behavior by means of states and transitions, and Alf, for specifying fine-grained actions. The state machine diagram describing the behavior of the type `ManagerR` is shown in the upper right corner of Fig. 2.

3 Research method

This study has been carried out by following the process shown in the activity diagram in Fig. 3. It can be divided into three main phases: planning, conducting, and documenting. This is in compliance with the well-established methodology for systematic studies [27,51]. Each phase was performed by one or more members of the research team conducting the study (see Appendix 10).

In order to address potential threats to validity and possible biases, we submitted the protocol describing the design of this study and the final report to external experts for independent review and criticism. Each review underwent a thorough refinement iteration of the design and reporting of this study.[7]

To allow easy replication and verification of our study, we made available to interested researchers a complete **replication package**.[8] It includes a description of the review protocol, the complete list of selected studies, extracted data, and the R scripts we developed for summarizing and synthesizing our findings.

In the following, we cover each phase of the process, highlighting the main activities and generated artifacts.
Planning The objective of this phase was to: (i) establish the need for a review of UML model execution strategies, (ii) identify the main research questions (see Sect. 3.1), and (iii) define the protocol to be followed by the research team.
Conducting In this phase, we performed the review itself by following all the activities defined in the review protocol:

- *Search and selection* we performed a combination of automatic search and snowballing for identifying a comprehensive set of potentially relevant solutions for the execution of UML models. With snowballing, performed after the automatic search, we aimed at enlarging the set of potentially relevant solutions by considering each previously selected research study and focusing on those articles either citing or cited by it [49]. Next, the candidate solutions were filtered in order to obtain the final list of primary studies to be considered in later activities of the review. Section 3.2 describes in detail the search and selection process.
- *Data extraction form definition* in this activity, we defined the set of parameters to be used for comparing primary studies based on the chosen research questions. This activity was systematically performed by applying a keywording process [39]. Section 3.3 describes the data extraction in detail.
- *Classification framework definition*: in this activity we organized the parameters pertaining to the technical characteristics of a solution for executing UML models. Because these were applicable to evaluating both tools and research studies, the result was a reusable classification framework, which can be used by both researchers and practitioners (see Sect. 3.3).

[7] We would like to thank the following external reviewers: Kai Petersen (Blekinge Institute of Technology), Daniel Sundmark (Mälardalen University), and Jérémie Tatibouet (CEA LIST Institute).

[8] http://cs.gssi.it/umlModelsExecution.

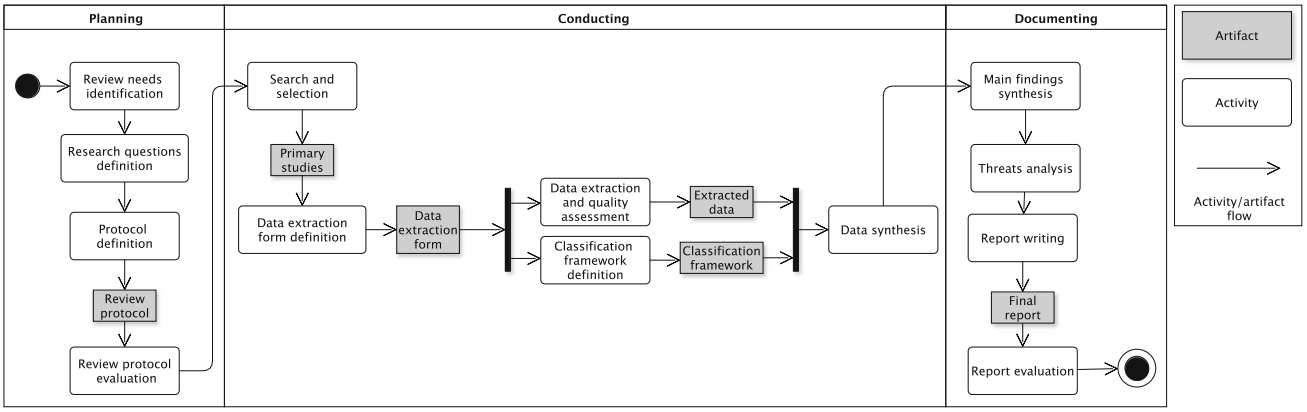


Fig. 3 Overview of the review process

- *Data extraction and quality assessment* in this activity we studied the details of each primary study, and based on that, filled in the corresponding data extraction form. Completed forms were then collected and aggregated for use in the next activity. More details about this activity are presented in Sect. 3.3. Furthermore, we assessed the quality of each selected research study according to a set of well-defined quality criteria (see Sect. 6.4).
- *Data synthesis* this activity focused on a comprehensive analysis and summary of the data extracted in the preceding activity. The main goal here was to elaborate on the extracted data in order to answer the research questions (see Sect. 3.1). This involved both quantitative and qualitative analysis of the extracted data. The details of this activity are presented in Sect. 3.4.

Documenting The main activities performed in this phase were: (i) a thorough elaboration of the data extracted in the preceding phase in order to properly position the obtained results in the appropriate context from both an academic and a pragmatic point of view, (ii) an analysis of possible threats to validity, especially those identified during the definition of the review protocol, and (iii) the writing of a technical report describing the performed study.

3.1 Goal and research questions

The goal of our study is to identify, classify, and evaluate the publication trends, characteristics, provided evidence, and limitations of existing solutions for the execution of UML models from a researcher's and practitioner's points of view.

This goal was then refined into the following research questions, each of which had a defined primary objective:

- *RQ1—What are the publication trends of research studies about solutions for the execution of UML models?*
  Objective: to identify and classify primary studies in order to assess interest, relevant venues, and publication types over the years.
- *RQ2—What are the technical characteristics of existing solutions for the execution of UML models?*
  Objective: to classify existing solutions for executing UML models (e.g., execution strategy used, which elements of UML are supported, application domains targeted), using a systematic classification framework (see Sect. 3.3).
- *RQ3—What evidence exists to justify adoption of an existing solution?*
  Objective: to investigate and qualify the quality of the evidence that could make a proposed solution applicable in practice. This is based on explicit quality criteria, such as the rigor and thoroughness of the validation strategies used (e.g., controlled experiment, industrial application, formal proofs).
- *RQ4—What are the limitations of existing solutions for the execution of UML models?*
  Objective: to identify current limitations with respect to the state of the art. In this context, we focused on the limitations (and thereby needs for improvement) of the solutions as they were reported in the analyzed primary studies.

3.2 Search and selection strategy

As shown in Fig. 4, our search and selection process was composed of two main sub-processes, each of which focused on a specific source of information:

- *Research studies search and selection* this covers the research described in peer-reviewed publications in various research-oriented venues and forums such as international journals, conferences, books, workshops;
- *Tools search and selection* this covers relevant open-source and commercial tools.

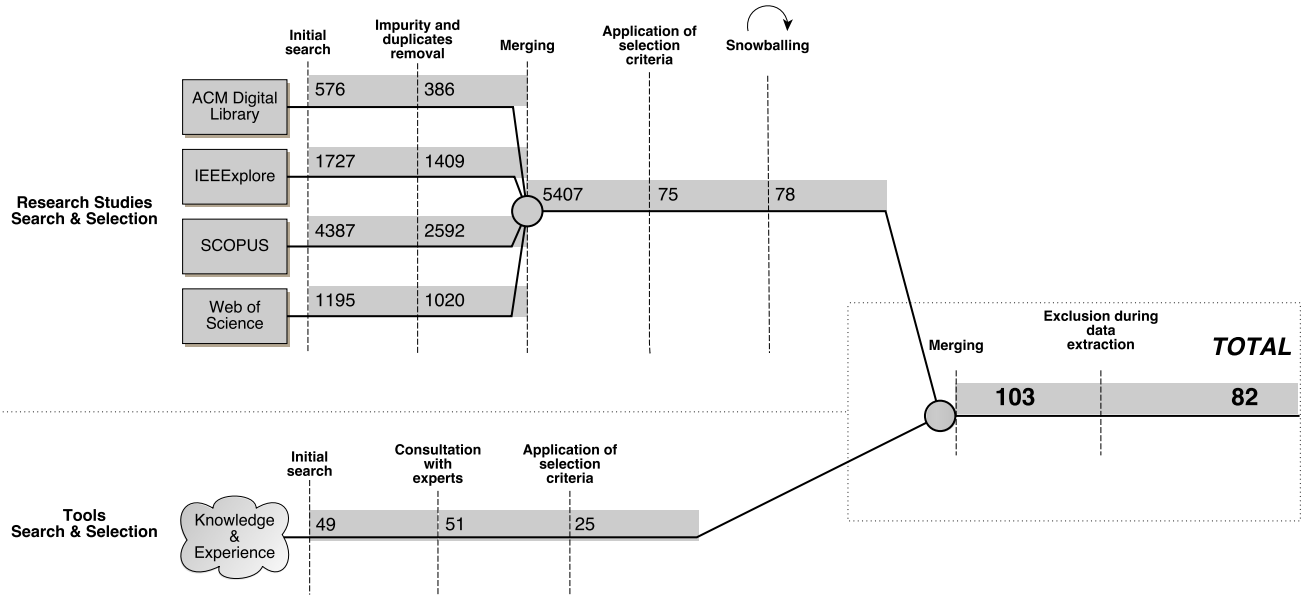


Fig. 4 Search and selection process

Both sub-processes included a stage in which each potentially relevant approach was evaluated against a set of selection criteria (see Sects. 3.2.2, 3.2.4). To handle those cases in a cost-effective way, we used the adaptive reading depth [39], because it was not necessary to read the full text of approaches that clearly did not qualify. Also, by following the method proposed in [6], each potentially relevant approach was classified by both the principal researcher[9] and the research methodologist as *relevant*, *uncertain*, or *irrelevant* according to the selection criteria described in Sect. 3.2.2. Approaches classified as *irrelevant* were immediately excluded, whereas all the other approaches were discussed, when needed, with the help of the study advisor.

In the following, we give a brief description of each stage of the two search and selection sub-processes.

3.2.1 Research studies search and selection

Before performing the actual search and selection of relevant research studies, we manually selected a set of pilot studies. These were selected based on the degree of domain expertise of the authors, as well as on an informal preliminary screening of available literature on the topic. The chosen pilot studies fully met our selection criteria (see Sect. 3.2.2) and are reported in the replication package of this study.

The pilot studies were used to validate and refine our search and selection strategy. We describe below the main stages of the research studies search and selection activity.

[9] The research team and the different roles are described in "Appendix A".

1. *Initial search* In this stage, we performed automatic searches on electronic databases and indexing systems [27]. As suggested in [27], to cover as much potentially relevant literature as possible, we chose four of the largest and most complete scientific databases and indexing systems in software engineering: IEEE Xplore Digital Library, ACM Digital Library, SCOPUS, and Web of Science.

```
uml
AND (execut* OR translat* OR compilat* OR interpret* OR synthes*
     OR simulat* OR debug* OR (code AND generat*))
AND (mde OR mda OR mdsd OR mdd OR model* OR diagram OR
     specification)
AND (software OR system OR tool)
```

Listing 1 Search string used for automatic research studies

To create the search string in Listing 3.2.1, we considered the research questions in Sect. 3.1 and the set of pilot studies. To ensure consistency, the search strings were always applied to the title, abstract and keywords of papers.

2. *Impurity and duplicates removal* Due to the nature of the electronic databases and indexing systems, the search results could also include items that were not research papers, such as conference and workshop proceedings, international standards, textbooks, editorials. Consequently, in this stage we: (i) manually removed these results in order to have a coherent set of potentially relevant research studies, and (ii) identified and removed duplicated entries.
3. *Merging* In this stage all relevant results from the different databases and indexing systems used in the first stage were combined.
4. *Application of the selection criteria* In this stage we considered all the selected studies and filtered them according to

a set of well-defined inclusion and exclusion criteria. These criteria are described in details in Sect. 3.2.2.

5. *Snowballing* To mitigate a potential bias with respect to the construct validity of the study, we complemented the previously described automatic search with a snowballing activity [22]. The main goal of this stage was to enlarge the set of potentially relevant studies by considering each study selected in the previous stages, and focusing on those papers that either cited or were cited by it. Technically, we performed a closed recursive backward and forward snowballing activity [50].

3.2.2 Selection criteria for research studies

We used the following *inclusion criteria* for the research studies:

- IS1 Studies proposing solutions for the execution of UML models.
- IS2 Studies in which UML models are: (i) the main input artifacts of the proposed execution method or technique, and (ii) executed without any manual intervention.
- IS3 Studies providing some kind of evaluation of the proposed solution (e.g., via formal analysis, controlled experiment, exploitation in industry, example usage).
- IS4 Studies subjected to peer review [51] (e.g., journal papers, book chapters, papers published as part of conference or workshop proceedings will be considered, whereas white papers will be discarded).
- IS5 Studies written in English.
- IS6 Studies for which the full text is available.

We used the following *exclusion criteria* for the research studies:

- ES1 Studies that do not provide any implementation of the proposed solution.
- ES2 Studies published before 1997 (because UML was adopted in 1997).
- ES3 Secondary and tertiary studies (e.g., systematic literature reviews, surveys).
- ES4 Studies in the form of tutorial papers, short papers, poster papers, editorials, because they do not provide enough information.

3.2.3 Tools search and selection

In parallel with the search and selection of research studies, we also performed a search and selection of **tools** used for executing UML models. The process we followed for searching and selecting relevant tools involved the following stages:

1. *Initial search* We based this stage on: (i) our personal experiences with execution of UML models, (ii) specific searches for querying generic web search engines with the search string defined in Listing 3.2.1, and (iii) exploiting knowledge garnered from existing networks of experts in the execution of UML models, e.g., by accessing forums, mailing lists.

2. *Inclusion of tools from research studies* During the search and selection for research studies we stumbled upon further tools used or referred by researchers. Those tools were included in the set of tools to be considered. If an included research study described the tool's characteristics for execution of UML models, no explicit tool entry is added to the set of tools (to avoid duplication).

3. *Consultation with experts* In this stage we consulted experts from both industry and academia to minimize the set of selected tools. For example, this required removing duplicates due to renaming of a tool over its lifetime.

4. *Application of the selection criteria* In this stage, we considered all the selected tools and filtered them according to well-defined inclusion and exclusion criteria. These criteria are presented in detail in Sect. 3.2.4. In this search and selection subprocess, we focused on tools and we collected information from associated documentation (e.g., white papers, user guides, Web pages), rather than from actual scientific papers.

3.2.4 Selection criteria for tools

The following tools *inclusion criteria* were used:

- IT1 Tools proposing methods or techniques for execution of UML models.
- IT2 Tools in which UML models are the main input artifacts of the proposed execution method or technique.

The following tool *exclusion criterion* was used:

- ET1 Tools leveraging third-party tools for executing UML models.

3.3 Data extraction and classification framework definition

The main goal of this activity was to create a **data extraction form** to be used to collect data extracted from each primary study. In our work, the data extraction form was composed of four facets, each of which addressed a specific research question (see Sect. 3.1). Specifically, for answering RQ1 (i.e., research question about publication trends), we considered standard information such as title, authors and publication details of each study. As for the other research questions, (i.e., RQ2, RQ3, RQ4), we followed a systematic process

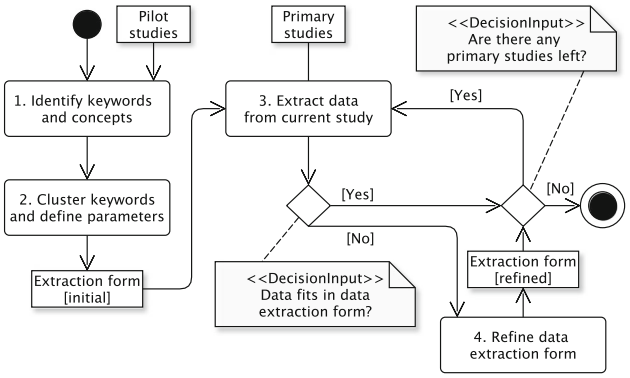


Fig. 5 Keywording and data extraction process

based on *keywording* for: (i) defining the parameters of each facet of the data extraction form, and (ii) extracting data from the primary studies accordingly. The goal of the keywording was to effectively develop an extraction form that could fit existing studies and tools, and which took their characteristics into account [39].

The activity diagram in Fig. 5 shows the process that we followed to define the data extraction form. The following describes the individual steps of this process:

1. *Identify keywords and concepts* We collected keywords and concepts by reading the full text of each pilot study.
2. *Cluster keywords and define parameters* We performed a clustering operation on collected keywords and concepts in order to organize them according to the identified characteristics. The clustering operation is very similar to the sorting phase of the grounded theory methodology [13]. More specifically, we considered all the keywords and concepts collected in the previous phase and iteratively grouped them together until a saturation of all the concepts has been achieved. The output of this stage was the initial data extraction form containing all the identified parameters, each of which represented a specific characteristic of a solution for executing UML models. The following steps were performed individually for each primary study.
3. *Extract data from current study* We first extracted information about the current primary study to be analyzed and then collected information based on the parameters of the data extraction form. Finally, we collected any kind of additional information that was deemed relevant but that did not fit within the data extraction form.
4. *Refine data extraction form* We reviewed the collected additional information. This could be due to any of the following:
   - the collected information was not interpreted correctly; in this case, the collected information was refined;
   - the parameters of the data extraction form were not representative enough for the considered primary study; in this case, the data extraction form was refined so that it better fit the collected information; previously analyzed primary studies were re-analyzed according to the refined data extraction form.

The above process was complete when all primary studies were analyzed. Once we had a complete data extraction form, we isolated the facet related to the technical characteristics of the approaches (i.e., the facet related to RQ2). This was then analyzed further to establish any relationships between the parameters, and also to determine the multiplicity and possible values of each parameter. This process was performed through several iterations and led to the definition of a **classification framework** for solutions for UML model execution. This classification framework is described in Sect. 5. It can be used by researchers and practitioners for classifying, comparing, and evaluating solutions for executing UML models in an objective manner.

3.4 Data synthesis

We designed and executed our data synthesis activity by following lessons learned and findings presented by Cruzes et al. [14]. Specifically, our data synthesis activity was divided into two main phases: vertical analysis and horizontal analysis.

In **vertical analysis** (cf. Sects. 4–7), we analyzed the extracted data to find trends and collect information about each parameter of our data extraction form. In this case, we applied the *line of argument* synthesis [51]. We present the results of this vertical analysis in Sects. 4, 5, 6, 7. Moreover, the complete set of extracted data for each primary study is given in "Appendix C".

For **horizontal analysis** (cf. Sect. 8), we analyzed the extracted data to explore possible relations across different parameters of our data extraction form. To that end, we cross-tabulated and grouped the data and made comparisons between two or more parameters of the data extraction form. The main goal of the horizontal analysis was to uncover the existence of possible interesting relations between data pertaining to different aspects of our research. For this purpose, we used contingency tables analysis as the strategy for evaluating the actual existence of those relations. The results of the horizontal analysis are presented in Sect. 8.

In both cases, we performed a combination of content analysis [18] (mainly for categorizing and coding approaches under broad thematic categories) and narrative synthesis [43] (mainly for detailed explanation and interpretation of the findings coming from the content analysis).

3.5 Threats to validity

In this section, we discuss the main threats to validity of our study and how we mitigated them.

External validity Concerning *research studies*, we employed a search strategy consisting of both automatic search and snowballing of selected studies. We use these two search strategies in combination to mitigate the threat to external validity. In both automatic search and snowballing, we did not consider gray literature (e.g., white papers) because in this phase we are focussing on the state of the art about UML models execution and peer-reviewed scientific venues are well-established publication targets for researchers. Nevertheless, given the quite high number of considered primary studies, this potential bias should not impact our study significantly. Concerning *tools*, we mitigated the external validity threat by applying a well-defined process for selecting relevant tools from the state of the practice. Namely, we performed (i) an initial search based on our direct experience, specific searches on generic web engines, and the knowledge from networks of experts, (ii) an analysis of the tools referred by selected research studies, and (iii) consultations with experts from both industry and academia.

Finally, in our search we considered only material written in the English language. This choice may potentially have led us to discard some primary studies published in other languages. However, the English language is the most widely used language for scientific articles and tools documentation, so this bias can be reasonably deemed to have a minimal impact.

Internal validity We mitigated this potential threat to validity by (i) rigorously defining the protocol of our study, and (ii) defining our data extraction form by following a well-defined and preliminarily piloted validated process (see Sect. 3.3). Regarding the validity of the synthesis of collected data, the threats are minimal, since we employed well-assessed descriptive statistics.

Construct validity We are reasonably confident that we have not missed any significant relevant research studies or tools. Specifically, *research studies* were searched by firstly applying an automatic search on multiple data sources. We considered multiple electronic databases to get relevant studies independently of publishers' policies (see Sect. 3.2). We are reasonably confident about the construction of the search string, since the terms used were identified by carefully following the research questions and by analyzing the set of pilot studies. Complementing the automatic search with snowballing increases our confidence in our search strategy. *Tools* were selected by applying the previously described search and selection process, involving our direct experience, web search engines, knowledge from existing networks of experts, selected research studies, and experts from both industry and academia.

After having collected all relevant research studies and tools, we rigorously screened them according to well-documented inclusion and exclusion criteria (see Sect. 3.2.1). This selection stage was performed by the principal researcher and the research methodologist. As suggested by Wohlin et al. [51], a random sample of 20 potentially relevant research studies and tools was identified and the inter-researcher agreement was measured using the Cohen–Kappa coefficient [12]. The results are very convincing, since we obtained a Cohen–Kappa coefficient of 0.80 for research studies and 0.91 for tools.[10]

Conclusion validity Firstly, we mitigated potential threats to conclusion validity by applying well-accepted systematic processes throughout our study and we documented all of them in our research protocol, so that this study can be replicated by other researchers interested in the topic. Secondly, the data extraction phase could have been another source of threats to the conclusion validity of our study, mainly because (i) other researchers may identify parameters and dimensions different from ours and (ii) our data extraction activity is based on human judgement about the solutions described in the primary studies. We mitigated this bias by (i) letting the parameters and their facets emerge from the pilot studies and refining them throughout the data extraction activity, (ii) performing an external evaluation by independent researchers who were not involved in our research, and (iii) having the data extraction process conducted by the principle researcher and validated by the secondary researcher.

4 Publication trends

In this section, we describe publication trends extracted from the analyzed research studies. More specifically, we aim at answering the following research question:

RQ1—What are the publication trends of research studies pertaining to solutions for the execution of UML models?

To answer RQ1, for each analyzed research study, we extracted the publication year and venue. The results obtained are discussed below.

4.1 Publication year

Figure 6 presents the distribution of publications on execution of UML models over time. From the collected data, we can observe that relatively few studies were published until 2007. Starting with 2008, we can see a growth in

[10] The result of the Cohen Kappa statistic is considered as *substantial* (*almost perfect*) when it is above 0.60 (0.80) [30].

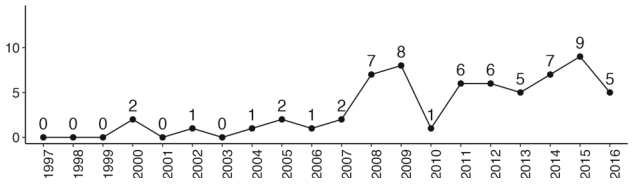


Fig. 6 Distribution of research studies by year

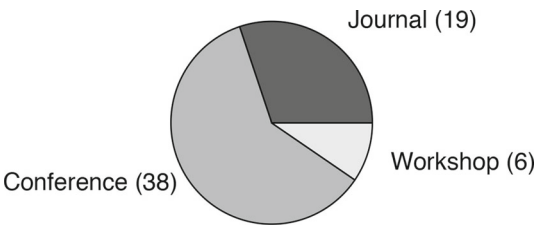


Fig. 7 Distribution of research studies by type of publication

the number of studies. Interestingly, this increase coincides with the publication of the first beta version of the fUML specification (in the OMG process, a "beta" specification is publicly released to encourage early implementation and also to identify possible issues which are handled by a Finalization Task Force, leading to a finalized specification that is formally published—in the case of fUML, this took until 2011). Indeed, the average number of publications between 1997 and 2007 is only 1 per year. Between 2008 and 2016, it reached a value of 6. This result confirms the growing focus on and increasing need for research in UML model execution. Given the steady rate of the number of publications per year over the past 8 years, we expect this trend to continue in the future.

The first research studies proposing a solution to executing UML models were published in 2000 (P62, P63). Specifically, the authors of (P62) presented a framework for generating simulation programs from UML models with the goal of predicting the performance of the system being modeled. In P63 they used the Fujaba environment to specify production control systems, generate Java code, and simulate their corresponding UML models.

We also analyzed the drop of the number of research studies in 2010, but did not find any clear reasons for it.

4.2 Publication venues

We classified the analyzed research studies to assess their distribution by (i) type of publication (i.e., journal, conference, or workshop paper) and (ii) targeted publication venues.

Figure 7 shows the publication types of the analyzed research studies. The most common publication type is a conference paper (38/63), followed by journal papers (19/63). The low number of workshop papers may be an indicator of the fact that, given the effort required for proposing a solution for UML model execution, researchers focusing on this research topic commonly target more scientifically rewarding publications (e.g., journal papers). Nevertheless, this is changing since there are now workshops and conferences fully dedicated to the topic, creating thereby a clearly defined dedicated research community. The first step toward this change is the International Workshop on Executable Modeling (EXE), co-located with the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), which has hosted three primary studies in 2015–2016.

Table 1 shows the publication venues that hosted more than one analyzed research study (the last row of the table is an aggregate of all the publication venues with only one research study). We can see that research on UML model execution is spread across a large number of venues (more than 50 in total), spanning different research areas such as embedded systems, reconfigurable systems, visual languages and human–computer interaction, distributed computing. Under this perspective, UML model execution is perceived today as orthogonal with respect to other research areas. A consequence is that researchers actually focus more on the benefits and effects of exploiting UML model execution (e.g., a gain in terms of system reliability or reconfigurability), rather than on the specifics of the execution mechanisms.

Highlights—RQ1. What are the publication trends of research studies related to solutions for the execution of UML models?

- From 2008 there has been a growing scientific interest on UML model execution; this positive trend has been more or less steady in the past 6 years;
- Conferences and journals are the most targeted publication venues, testifying that UML model execution is becoming a significant research theme;
- Research on UML model execution is spread across a large number of heterogeneous venues, with researchers focusing more on benefits and effects of model execution, rather than on the specific execution techniques;

5 Technical characteristics

In this section, we describe the technical characteristics of the analyzed solutions for executing UML models. More specifically, it aims at answering the following research question:

RQ2—What are the characteristics of existing solutions for the execution of UML models?

Table 1 Publication venues with more than one primary research study

Publication venue	Type	#Studies
European Conference on Modelling Foundations and Applications (ECMFA)	Conf.	3
Design, Automation and Test in Europe Conference and Exhibition (DATE)	Conf.	3
Design Automation for Embedded Systems (DAES)	Conf.	3
Software and System Modeling (SoSyM)	Journal	3
ACM Symposium on Applied Computing (SAC)	Conf.	2
Forum on Specification and Design Languages (FDL)	Conf.	2
Brasilian Symposium on Computing System Engineering (SBESC)	Conf.	2
Executable Modeling (EXE)	Work.	3
Others	–	42
Total	–	63

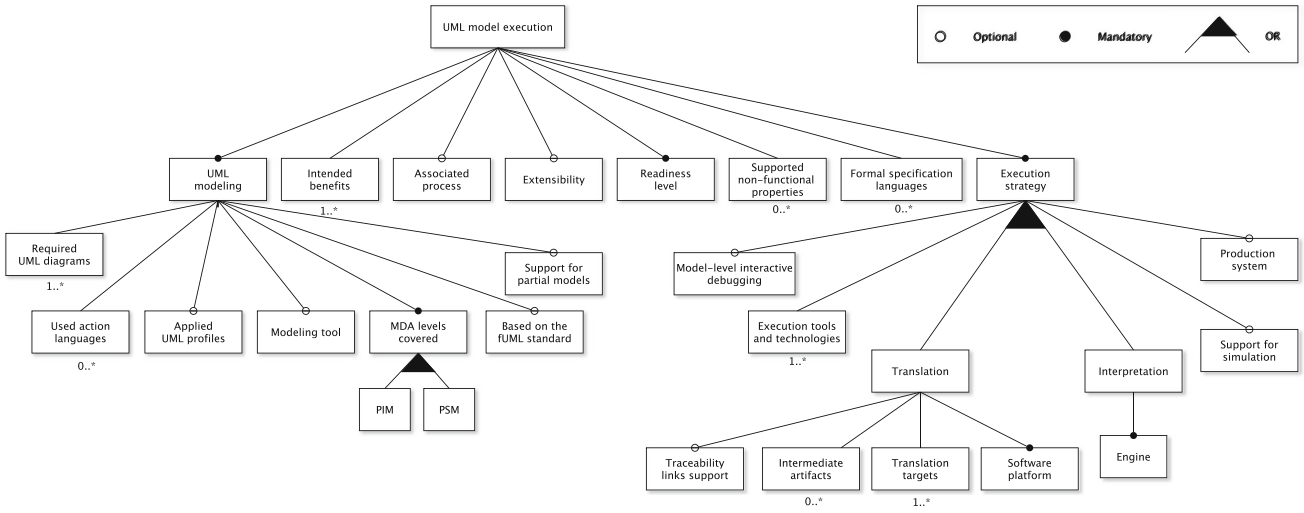


Fig. 8 Classification framework of solutions for UML model execution

Figure 8 presents the classification framework we defined for answering this research question. Note that this classification framework was obtained through a systematically conducted process (i.e., the keywording activity for the data extraction form), and it was applied to all the 80 primary studies of this study, which strongly suggests that it provides a general facility for evaluating and comparing approaches for execution of UML models. For reasons of brevity, we do not provide a fully detailed description of all the parameters of the classification framework; however, we do briefly elaborate on each of them when discussing the results of our study in the remainder of this section.[11]

5.1 UML modeling

In this section we examine the characteristics of the analyzed solutions with respect to the way they model a software system with UML.

[11] The interested reader can refer to our "replication" package for a thorough and extensive discussion of all characteristics of our classification framework.

Required UML diagrams This parameter represents the non-empty set of UML diagrams required by the analyzed solution to be used when modeling a software system (Fig. 9a). Class (48/82), state machine (36/82), and activity (33/82) diagrams represent the most commonly required model representations, often in combination. Overall, the expected trend is that both structural and behavioral models are used when defining executable models.

Used action languages To make UML models executable, in addition to well-defined execution semantics, further languages (i.e., action languages) may be needed for specifying fine-grained data and behaviors [36]. This parameter specifies which action languages were used by the analyzed solutions. Figure 9b shows that there is a remarkable number of solutions (22/82) which do not use action languages. It is interesting to note that only 1 tool (out of 19) does not use any form of action language, whereas the number of research studies not using action languages is much higher (21/63).

Almost half of the solutions using an action language (37/60) leverage UML-compliant languages, but only a small portion employs the current standard, Alf (9/37). Most solu-

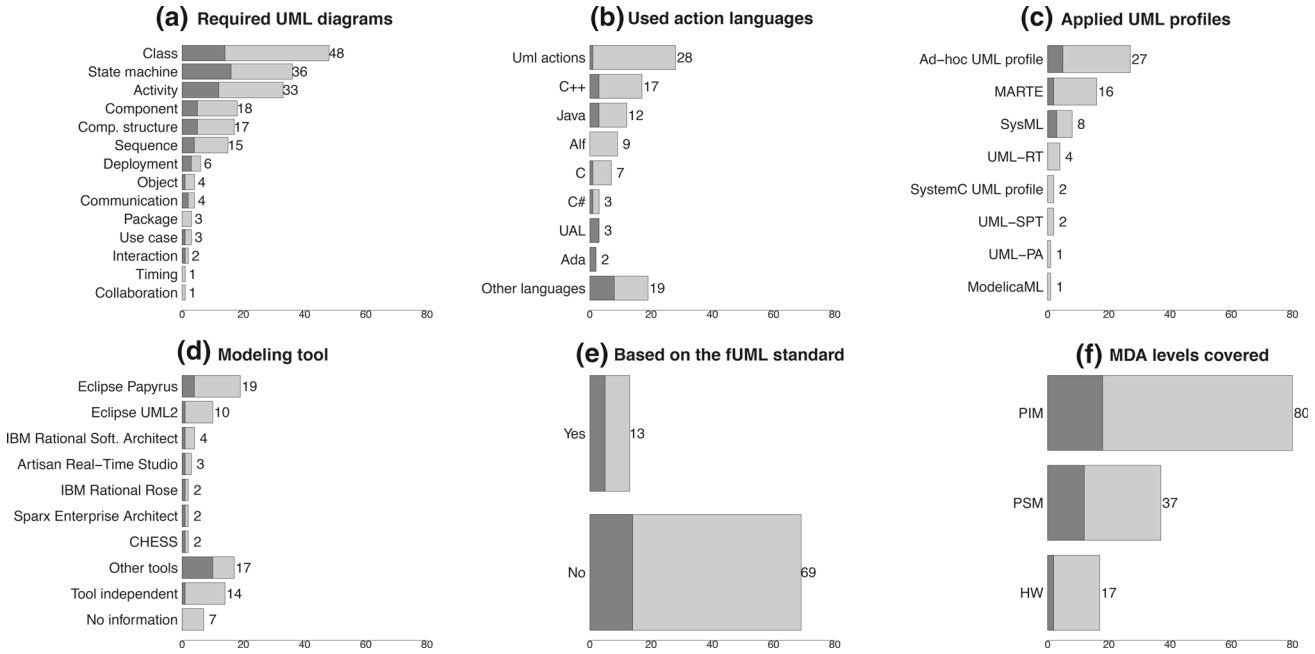


Fig. 9 UML modeling—analysis results (light bars: research studies, dark bars: tools)

tions (46/60) leverage popular programming languages as action languages, although these are not semantically consistent with UML.

Applied UML profiles With this parameter, we describe whether an analyzed solution relies on any UML profile to enable execution. Note that standard UML is not itself executable, due to the presence of numerous variability points and unspecified details (even fUML, a standardized executable subset of UML, has variability points). Hence, the execution of UML models requires additional semantic details to be provided. This may be done explicitly, using a suitable UML profile that defines the necessary semantics (as in 49/82 of our primary studies), or implicitly, by solution-specific semantics built into the model transformation or execution technologies. With this parameter, we identify the first case.

Among the solutions using UML profiles (see Fig. 9c), the most commonly used standard profile is MARTE (16/49). It is interesting to note that, in a relevant number of cases (27/49), a solution-specific profile was explicitly defined.

Modeling tool This parameter specifies the modeling tools which are used by the analyzed solution for UML modeling (see Fig. 9d). The most used tool appears to be Papyrus[12] (19/82). The number of tool-independent solutions (14/82) is fairly low, it is worth noting that more than half of the solutions (51/82) leverage open-source tools.[13]

Based on the fUML standard fUML defines an execution semantics for a large subset of standard UML. This parameter identifies whether the analyzed solution conforms to the fUML specification. Different solutions may leverage different implementations as long as they conform to the specification. Among the analyzed solutions (see Fig. 9e), the majority does not conform to the semantics defined by fUML. However, it should be kept in mind that fUML was not available until 2011. If we consider the studies proposed after 2011 (37/82), we note that more than one-third (13/37) is based on fUML. Note that, in order to address the informality of UML, in approaches not based on fUML, execution semantics is enforced at model transformation level, e.g., by using 3GLs as action code in the models.

Covered MDA levels Inspired by the well-known MDA [29] modeling levels, this parameter specifies whether the analyzed solution covers the platform-independent (PIM) and/or the platform-specific (PSM) modeling levels (see Figs. 9f, 10). While the majority (80/82) exploits models at the PIM level, only very few solutions (17/82) actually model the hardware platform in detail (HW). A significant number of solutions exploit both PIM and PSM models (20/82), or even combine PIM, PSM, and HW (16/82).

Support for partial models This parameter specifies whether the analyzed solution supports the execution of models with elements that are tentative, missing or not well-formed [53]. This capability is particularly useful in early design phases where different design approaches are proposed and evaluated. By supporting the execution of partial models, it is possible to (1) significantly reduce the amount of time and

[12] http://www.eclipse.org/papyrus/.

[13] This should not be misinterpreted to mean that these tools dominate in terms of usage volume in industrial practice, where commercial tools still seem to be dominant.

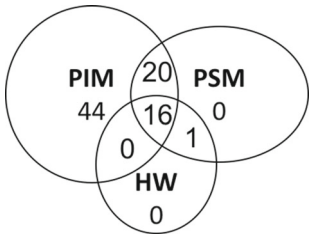


Fig. 10 Distribution of solutions according to the MDA levels

effort required to evaluate an approach and (2) increase the likelihood that unpromising approaches will be identified and discarded early in the design cycle. The analysis revealed that, to date, only one solution for execution of UML models explicitly provides support for the execution of partial models; in P10, the authors provide semantics for elements that are "tentative, missing or not well-formed" so that they can support the execution of models that are still under development.

5.2 Execution strategy

In this section, we provide the characteristics of the analyzed solutions based whether they execute UML models through translation or interpretation,[14] as well as a set of parameters independent of the execution type. We start with the set of independent parameters and continue with parameters specific to translational and interpretive execution.

Execution tools and technologies This parameter refers to all third-party tools and technologies used by the analyzed solutions to execute UML models (e.g., programming languages, model transformation languages). From our analysis (see Fig. 12d), we discovered that a substantial number of the solutions (40/82) employs Java as execution technology. The use of Java varies from model transformations to execution in a simulated environment. Note that most tools (16/19) heavily exploit Java, while most academic solutions (41/63) leverage technologies specifically developed for model manipulation (i.e., model transformation languages). Among them, QVT,[15] Acceleo,[16] and ATL.[17]

Model-level interactive debugging This parameter specifies whether the analyzed solution provides some level of support for interactive debugging at the level of the source UML model (e.g., support for specifying breakpoints in the models, step-by-step execution at the UML level.). As shown in Fig. 12e, only a few solutions (21/82, of which 13 are tools) provide such a feature, which suggests that model execution is (too) rarely a way to assess the model itself.

Support for simulation This parameter indicates whether the analyzed solution supports simulation of UML models. Simulation is defined as execution of a model in an environment (e.g., an IDE) that is different from the ultimate intended target environment (e.g., a controller board). This can be useful for a number of reasons, including the unavailability of the target environment, the availability of tools that are not present or not available in the target environment (e.g., debugging tools), or other pragmatic or economical reasons. Among the analyzed solutions, more than half (49/82) provide support for model simulation (see Fig. 12f).

It is important to note that in certain cases (such as in tools like iUML or BridgePoint) interpretive and translational approaches live under the same hood, but they are provided through different mechanisms and for different purposes; for example, simulation may be used for models validation/analysis/debugging, whereas model execution is used for running the system on the target platform. Our analysis reveals that among the 14 solutions adopting an interpretive execution strategy, 11 of them also support models simulation. In other words, even if we consider the support for simulation and the interpretive execution strategy as disjoint aspects of analyzed solutions, in the majority of the cases solutions supporting UML models simulation provide also simulation support.

Production system This Boolean parameter identifies whether source models are executed on the ultimate target platform (e.g., full code generation and execution), or not (e.g., models simulated/run in the modeling tool only). The significant number of solutions (29/82) that do not provide execution on target platform,(see Fig. 12g), suggests that model execution is considered beneficial for early design assessment too.

5.2.1 Translational execution

Overall, Fig. 11.a shows that the number of translational solutions (70/82) clearly outnumbers the number of interpretive (14/82) solutions. From these numbers, we can conclude that the landscape of solutions applicable in real-world scenarios for the execution of UML models is dominated by translational approaches. This shows a trend of the community in choosing the pragmatic solution represented by translational execution, resulting in a myriad of repetitive solutions answering the same questions, which, considering the relevant number of research studies and tools on the topic, are not definitively answered yet.

Translation targets This parameter describes the target programming languages of the translated source models (see Fig. 12a). A total of 43 different languages are targeted, with the most common being Java (26/82). We found out that 3 solutions translate UML models directly to object code. Two of them are hybrid, since they translate UML models to a for-

[14] Note that one approach can provide both translational and interpretive mechanisms (e.g., for code generation and model simulation, respectively).

[15] http://www.omg.org/spec/QVT/.

[16] https://eclipse.org/acceleo/.

[17] https://wiki.eclipse.org/MMT/ATL_Transformation_Language_(ATL).

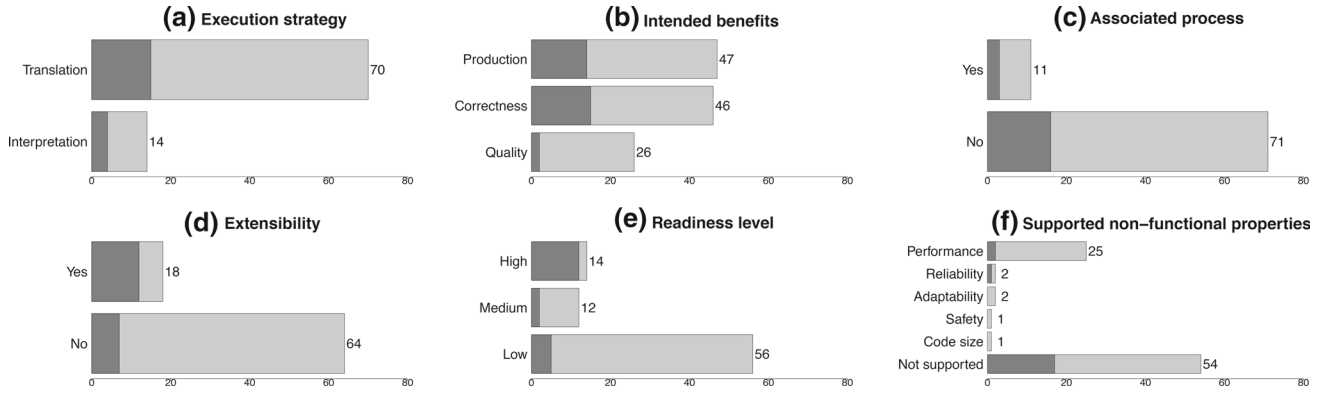


Fig. 11 UML model execution solutions—analysis results (light bars: research studies, dark bars: tools)

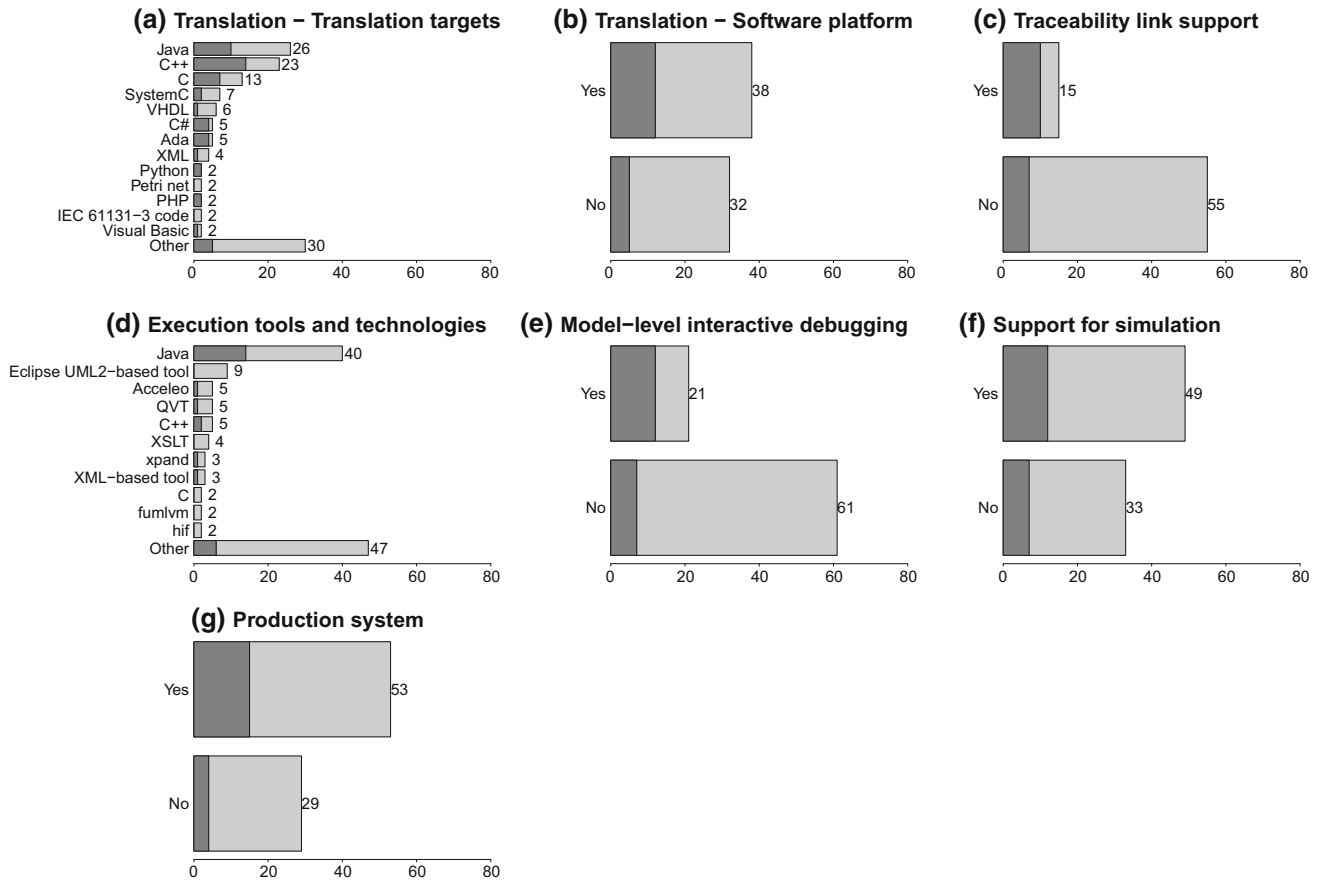


Fig. 12 UML model execution strategies—analysis results (light bars: research studies, dark bars: tools)

mat that is interpreted by ad hoc virtual machines. The three solutions leverage very limited subsets of UML; this makes them only suitable for limited applications and can explain their negligible impact in the community. It is interesting to note that direct translation to object code is not provided in any of the tools that we analyzed. Each solution employed a specific compiler producing executables for a specific target platform. No solutions used the same compiler nor targeted the same platform.

More specifically, the well-known open-source Gnu Collection Compiler (GCC) was used in P30 for generating a simplified form of abstract syntax tree, which was then used for GCC optimizations. By following a different line of reasoning, the authors of P55 realized a model compiler, which

takes as input UML models and compiles them into assembly code, from scratch. The produced assembly code can then be executed by a custom virtual machine called UML Virtual Machine (UVM). In P58, an ad hoc compiler and target platform were developed as well, where *UML specifications are compiled to equivalent binary representations, which are directly executed on [an ad hoc] Abstract Execution Platform (AEP).*

Software platform This Boolean parameter specifies whether the generated executable is meant to run on a specific software platform (e.g., OS, user-defined middleware, runtime libraries). The landscape of solutions is balanced with respect to this parameter (see Fig. 12b).

Traceability links support This parameter represents whether the analyzed solution provides mechanisms for generating explicit traceability links [4] connecting the UML models and other artifacts being executed (e.g., the source code generated by translation, the compiled binaries). Such links are crucial for certification in safety-critical domains, and are useful for a variety of purposes, including: debugging in cases where errors are encountered during execution, errors during compilation of the generated code, or for back propagation from the execution to models [10].

Fig. 12c we can see that only a small set of solutions (15/82), mostly tools (12/15), provide traceability links. A consequence of this is that these solutions are not well suited, for example, in situations where there are needs for vertical navigation from models to code and back based on precise correspondences. An example could be execution-based model assessment and optimization.

5.2.2 Interpretive execution

By "interpretive" execution we mean that the UML model is interpreted to be run on the designated platform. The parameter is composed of the name of the interpretation engine used for interpreting and executing the UML models. The overall number of interpretive solutions was 14. Among them, none seems to provide a solution for the execution of UML models on the actual target platform. Instead, they focus on higher-level execution for simulation and model-based analysis. Regarding adopted interpretation engines, none stood out as the preferred one. (See Table 2.)

Table 2 Interpretation engines

Engine name	Studies
Moka—Papyrus	P18, T14
fUML virtual machine	P24, P25
Ad hoc engine based on IBM RSA-RT	P31
fUML engine based on Kermeta	P32
UML interpreter engine	P34
MOCAS	P23
Populo	P48
ACTi	P49
fUML reference implementation	T13
iUML	T11
BridgePoint	T12

5.3 Intended benefits

This parameter identifies the benefits that the analyzed solution is intended to support explicitly. The possible values are: *correctness*, meant as improving functional correctness of models (e.g., through model-based analysis), *quality*, meant as improving non-functional aspects of the generated executable artifact (e.g., through optimized model compilation), or *production*, meant as reducing the effort needed for producing executable artefacts from models (e.g., through automatic code generation). As shown in Fig. 11b, in the majority of the cases solutions aim at decreasing production effort (47/82) or at improving model correctness (46/82). While most of the tools (13/19) focus on production, research studies display a fairly even distribution among the three benefits (29/63 on correctness, 34/63 on production, 24/63 on quality).

5.4 Associated process

This parameter specifies whether the proposed solution is closely associated with a specific development methodology. As shown in Fig. 11c, from our analysis we can notice that only a few solutions (11/82) explicitly position themselves within a specific methodology. The only well-known methodology is Shlaer–Mellor's, associated to the tools iUML (T11) and BridgePoint (T12). Other ad hoc methodologies are CHESS (T4, P6), MADES (P13), GenERTiCA (P5), MaCMAS (P54), and those unnamed provided in P4, P14, P21, and P60.

5.5 Extensibility

This parameter shows whether the proposed execution mechanisms can be extended or customized with additional components and capabilities, including those provided by third parties (e.g., plug-in based approaches, support for ecosystems of third-party modules). As shown in Fig. 11d, proportionally, only a minimal part of research studies (5/63) consider such capabilities, while more than half of the tools (12/19) provide extensibility mechanisms.

5.6 Readiness level

The Technology Readiness Level (TRL[18]) is proposed as a systematic metric/measurement system for assessing the maturity of a particular technology, and is defined as an integer n where $1 \leq n \leq 9$. Consequently, the readiness level of an analyzed solution and can be one of: *LOW* if $n \leq 4$ (i.e., if the solution was either formulated, validated or demonstrated at most in lab), *MEDIUM* if $5 \leq n \leq 6$ (i.e., if the solution was either validated or demonstrated in the relevant environment), and *HIGH* if $n \geq 7$ (i.e., if the solution was either completed, demonstrated, or proven in operational environment). We assigned a TRL value depending on how the solution was validated. It is important to note that the TRL parameter refers to the whole proposed solution (e.g., the realization of the whole tool chain, if available) and on how it has been evaluated (e.g., its feasibility has been tested via a simple example, or it has been validated in an operational environment).

As shown in Fig. 11e, it turns out that most of the analyzed solutions (56/82) have *LOW* readiness level. Only two research studies (P3, P56) have *HIGH*, while the remaining (12/82) are tools.

5.7 Supported non-functional properties

This parameter represents the set of non-functional properties recognized by the analyzed solution. By "recognized" we mean that the solution provides explicit mechanisms to assess, optimize, and/or reason about those non-functional properties (e.g., the solution supports analysis of the energy consumed by the system, or its performance, security.). As shown in Fig. 11f, more than half of the analyzed solutions (54/82) do not focus on reasoning on specific non-functional properties. Among the ones which do, performance is the most addressed property (25/28).

5.8 Formal specification languages

This parameter displays the formal specification languages (see Table 3) eventually used in conjunction with the UML models for, e.g., proving security properties, formally verify services composition. Formal language specifications are in some cases generated from UML models, and in others defined along with them; nevertheless, primary studies did not report enough details to be able to deeply categorize the two variants. Most solutions (69/82) do not employ any formal language. Among those that use formal languages, the majority performs model checking.

Highlights—RQ2. What are the technical characteristics of existing solutions for the execution of UML models?

- Solutions providing translational execution clearly outnumber interpretive solutions. Interpretive solutions are mainly addressing higher-level execution (e.g., for simulation). Solutions for translation to object code only leverage very limited subsets of UML;
- Class, activity, and state machine are the most used diagrams;
- Most solutions employ 3GLs as action languages, while only very few adopted the standardized action language for UML (Alf);
- Currently, there is only one solution which provides support for execution of partial models;
- Out of the analyzed translational solutions, only a very few provide explicit traceability links, which can be exploited for consistent navigations between models and generated code;
- Very few solutions provide support for model-level interactive debugging (note that on the other hand most tools provide this feature);
- Among the very few solutions that display *HIGH* as readiness level only two are research studies;
- A very small amount of solutions explicitly provides mechanisms which enable extension and customization.

6 Provided evidence

In this section, we describe the evidence provided by the analyzed research studies (52/70, since tools are excluded). More specifically, it aims at answering the following research question:

RQ3—What is the evidence that motivates the adoption of existing solutions for the execution of UML models?

Intuitively, the goal of answering RQ3 is to assess what is the potential for industrial adoption of research studies on UML model execution today. The underlying assumption of RQ3 is that research studies which have been (rigorously) evaluated in industry and with higher quality are more likely to be adopted in industry.

[18] The TRL measurement system is employed by the Horizon 2020 European commission for the work program of 2014/2015: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf.

Table 3 Used formal specification languages

Formal language	Usage	Study
Petri Net [41]	Translation target for performance analysis	P38
	Translation target for acting as reference for services orchestration	P54
NuSMV [11]	Generated for formally verifying the correctness of the specification	P17
	Translation target for verifying temporal properties	P56
FFSM (ad hoc)	Generated on the fly and interpreted at runtime	P31
	Translation target for execution and analysis	P6
KIV [21]	Automatically generated for proving security properties	P44
Jolie [38]	Translation target for formal verification of services composition	P53
Lisp [46]	Translation target for verification of user-defined properties and simulation	P57
BHDL [7]	Intermediate representation during the translation and properties verification	P60
Event B [2]	Action language in UML models and intermediate representation during the translation for properties verification	P60
Z [42]	Formal verification of user-defined properties	P61
TML (ad hoc)	Translation target for performance, liveness, reachability, and scheduling analysis	P2
ForSyDe (ad hoc)	Generated for formally verifying the behavioral of the modeled system	P10

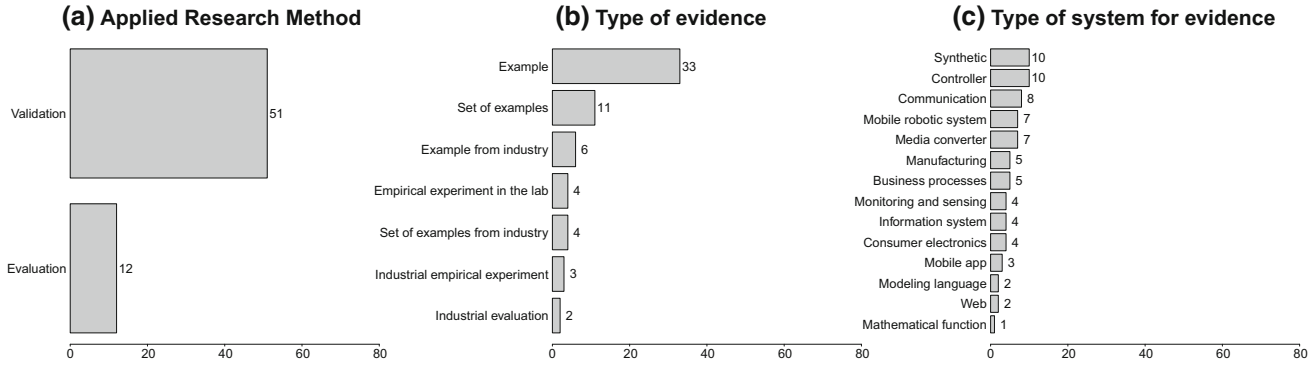


Fig. 13 Provided evidence and identified limitations—analysis results

6.1 Applied research method

This parameter represents the type of applied research method used to assess the analyzed solution. Based on [40, fig. 19], the possible values of this parameter are *validation* and *evaluation*. *Validation* is done in lab, whereas *evaluation* takes place in real-world (industrial) contexts. The latter generally provides a higher level of evidence about the practical applicability of a proposed solution, provided that the considered control groups are sufficiently diverse and independent, and that the number of participants in the studies is sufficiently large. Our analysis (see Fig. 13a) uncovered that only a very small number of the analyzed research studies (12/63) provided an *evaluation*.

6.2 Type of evidence

This parameter describes the type of evidence (e.g., example, empirical study) exploited by the analyzed research study for the purposes of validation and/or evaluation. The different types of evidence are:

- Example: 1 in-house example.
- Set of examples: several in-house (not industrial) examples, several runs and comparisons of one example, even several models for the system, e.g., with different sizes.
- Empirical laboratory: case studies, controlled experiments and empirical evaluations in laboratory.
- Industrial example: example coming from industry and performed in laboratory.
- Set of industrial examples: several industrial examples (even several models for the system, e.g., with different sizes).
- Empirical industrial: case studies, controlled experiments and empirical evaluations either in industry or in real-world scenarios.
- Industrial evaluation: evaluation performed by industrial actors, solution used in industry.

Table 4 Quality assessment criteria

ID	Quality criteria
$Q1$	Is there a clear statement of the *aims of the research*?
$Q2$	Is there an adequate description of the *context* (e.g., industrial use, laboratory-based investigation, product) in which the research was carried out?
$Q3$	Is there an adequate justification and description of the *research design*?
$Q4$	Is there a clear statement of *obtained findings*, including data supporting them?
$Q5$	Is there a critical discussion of the *researchers' roles, potential bias, and influence* on the proposed research?
$Q6$	Is there a critical discussion of potential *limitations* of the proposed research?

Figure 13b shows that the majority (44/63) of the analyzed solutions rely on the least systematic and least reliable typology, that is, on examples. Of these, only a few (11/44) use multiple different examples.

Out of the few that provide evidence in industrial settings (15/63), only three rely on empirical evaluations, whereas the majority of them (10/15) apply the proposed solutions on examples coming from industry. Only one research study (P6) actually performed a full-fledged industrial evaluation of the proposed solution. These results related to industrial-based evidence are actually making evident the perception that academic results on UML model execution are not fully transferred to or at least evaluated by industry. This can be considered as an interesting research gap to be explored in the future.

The obtained results also show that more than half of the research studies (39/63) present the application of their proposed solution using only a single example, independently of whether the example comes from industry or not. However, reaching conclusions about a complex topic like a solution for UML model execution from a single (usually simplified) example is dubious; this may negatively affect the potential impact that a proposed solution may have. The research community can address this potential issue by evaluating their proposed solutions on a set of complementary examples or applications in order to provide different perspectives over the proposed solutions.

6.3 Type of systems for evidence

This parameter describes the type of system used for providing evidence. Figure 13c presents the distribution of the analyzed studies with respect to the type of system used in obtaining evidence. The obtained results are spread across a variety of alternatives, ranging from communication applications, consumer electronics, mobile apps. This leads to the conclusion that there is no significant preference when it comes to the type of system used for providing evidence.

Interestingly, one of the two most used types of system is the *synthetic* one (10/63), meaning that the proposed solution has been applied to an artificial ad hoc example which is not grounded in any specific real system. The other most used type is *controller* (10/63). *Communication systems* (8/63), *manufacturing systems* (5/63), and *information systems*(4/63) are some of the types utilized (the interested reader can refer to the figure for the complete list).

Finally, we can notice that recent technological trends are reflected in the types of systems, such as *mobile robotic systems* (7/63) and *mobile apps* (3/63). This shows that UML can be successfully used for modeling and executing modern software systems.

6.4 Quality assessment results

In the context of this research, we assessed the quality of each selected research study with the objectives to: (i) provide an indication of the overall level of quality of academic research on executing UML models, and (ii) complement our findings about the evidence that motivates the adoption of solutions for executing UML models (RQ3).

We defined a set of criteria for assessing the quality of research studies in an objective and unbiased manner. We decided to follow the strategies already applied in [5,19], and we based our quality assessment strategy on the assessment instrument used in [17]. Specifically, we formulated the quality score of a study according to a set of criteria formulated as questions. Table 4 presents the questions we used for our study, which were inspired by those proposed in [5,19]. Each quality assessment question was answered by assigning a numerical value (1 = "yes", 0 = "no", and 0.5 = "to some extent") [19].

It is important to note that the quality criteria reported in Table 4 focus more on the quality of the paper (e.g., its clarity, rigor, precision in describing aspects of the performed research.) rather than on the quality of the research itself. The quality of research studies is important because a badly reported scientific article may suffer in terms of impact and

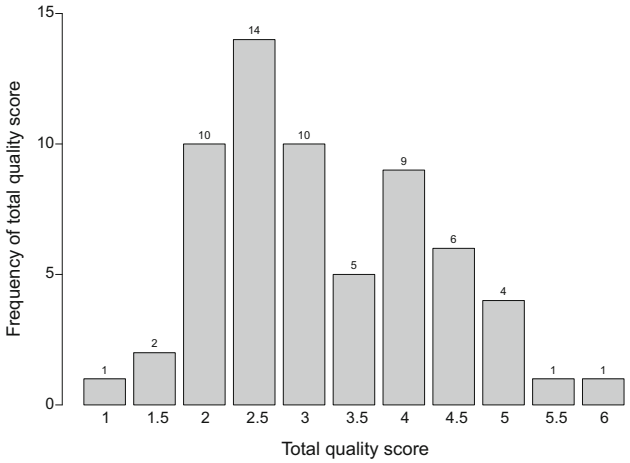


Fig. 14 Quality assessment—analysis results

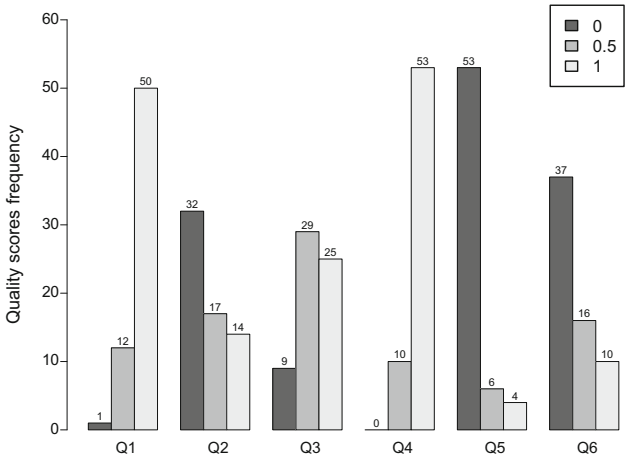


Fig. 15 Quality assessment—criteria-specific analysis results

transfer to industry, mainly because it may be difficult to locate relevant information in the study itself and important information may be missing.

Figure 14 shows the frequency of the total quality scores achieved by the selected research studies. The maximum score for a study is 6, i.e., scoring 1 for each question of the quality criteria. By looking at the obtained data we can tell that the majority of the studies achieved a score between 2 and 4, with only three studies falling below a score of 2, and 12 studies scoring above 4. The results tell us that the majority of the studies performed well, even though they still left something to be desired from a research quality point of view (e.g., many studies failed in describing precisely the context of their research and its potential limitations).

Figure 15 shows the frequencies of the scores of our selected research studies over the six quality criteria. In this context, the good news is that the majority of research studies performed well when providing a clear statement of the aims of their research ($Q1$) and the obtained results ($Q4$). Also, the justification and description of the research design ($Q3$) were described fairly well, with 25 studies doing it fully, 29 doing it partially, and 9 that did not report them at all. However, we can see insufficient scores with respect to the adequacy of the description of the context in which the research was carried out ($Q2$), with 32 studies not even saying anything about it. A similar situation can be seen for the reporting of a critical discussion of the researchers' roles, their biases and influence on their studies ($Q5$), and of the potential limitations of the proposed research ($Q6$). Overall, this analysis reveals the main issues pertaining to the quality of documenting current research on execution of UML models. Clearly, the research community must do better on these aspects of their work.

It is important to note that one of the reasons for having lower quality scores may be related to the page limits of venues where the primary studies have been submitted. In order to better investigate on this aspect, we built a contingency table between each single quality criteria $Q1$–$Q6$ and the type of publication (i.e., journal, conference, workshop). The analysis of the contingency tables reveals that the type of publication has a slight influence on the $Q2$ and $Q3$ quality criteria, where we noticed that journal papers have higher quality (i.e., a score of 1 for $Q2$ and $Q3$) with respect to overall trends. In contrast, the publication type does not play a significant role when considering $Q1$, $Q4$, $Q5$, $Q6$, and their aggregation computed by summing all $Q1$–$Q6$ criteria.

Based on the results obtained, in Table 5, we list the research studies with a quality score higher than or equal to 5. They can serve as good examples of high-quality research that can inspire future research studies in the area.

Highlights—RQ3. What is the evidence that motivates the adoption of existing solutions for the execution of UML models?

- The majority of the analyzed studies provide validation rather than evaluation (according to the definitions in [40, fig. 19]);
- A small number of studies provides evidence by experimentation in industrial settings; among them, only a few rely on empirical evaluation.
- The majority of the research studies are of good quality, although many of them fail in describing precisely the context of their research, the researchers' roles, and potential limitations.

Table 5 Research studies with quality score equal or higher than 4

ID	Title	Year	Score
P23	Environment modeling and simulation for automated testing of soft real-time embedded software	2013	6.0
P6	On the Generation of Full-Fledged Code from UML Profiles and Alf for Complex Systems	2015	5.5
P27	A Plug-in Based Approach for UML Model Simulation	2012	5.0
P56	FSMC+, a tool for the generation of Java code from statecharts	2007	5.0
P61	Deriving executable process descriptions from UML	2002	5.0
P3	On the automated translational execution of the action language for foundational UML	2016	5.0

7 Identified limitations

This section describes the limitations (and thereby need for improvement) of the solutions as they were reported in the analyzed research studies. More specifically, it aims at answering the following research question:

RQ4—What are the limitations of existing solutions for the execution of UML models?

Table 6 shows the distribution of the identified limitations and needs for improvement across our primary studies. Among them, the most common type of needed improvement is *expressiveness enhancement* with respect to the coverage of UML concepts (31/63); this result is not surprising, given the huge number of concepts of the UML metamodel; indeed, as of today the UML metamodel contains 264 classifiers (including abstract classifiers) such as *Class*, *Object*, *Activity*, *Transition*. It is understandable that many research teams do not have the resources for providing a complete approach with respect to the full set of concepts related to specific UML diagrams since they may be hundreds. In this context, an interesting perspective is provided by the potential added value that a dedicated company or a university spin-off may give to provide more complete tool support. An example of this positive synergy is the Eclipse Papyrus project, which started as an open-source project lead by the LISE team of a research division of the French alternative energies and atomic energy commission (CEA) and is supported by an industry consortium under the Eclipse Foundation umbrella now.[19]

Along the same lines, another recurrent improvement among research studies is *tool enhancement* (20/63). Similarly to the case of expressiveness, we can trace the motivation behind this result to the limited resources of researchers when considering a large language like UML. Also, it is important to note that in some cases research groups may be more interested in providing a prototype tool for giving evidence about their scientific results, rather than in providing an industry-ready, fully functional tool. As a natural consequence of this situation, tool enhancement for a specific research-driven solution is one of the most common needs for improvement. Other relevant possible improvements are additional analysis at model level (12/63) and additional evaluation (11/63).

In order to better put into context the identified limitations, we performed a detailed analysis about the potential correlations between identified limitations and all the other parameters of our classification framework; the results of this analysis are presented in Sects. 8.13 to 8.16.

Highlights—RQ4. What are the limitations of existing solutions for the execution of UML models?

- The most common limitation is represented by the supported expressiveness in terms of covered UML concepts.
- Another relevant limitation is related to tool enhancement.

8 Horizontal analysis

The goal of our horizontal analysis is to investigate on possible correlations between related parameters of the classification framework. To that end, while designing the classification framework, we kept track of potentially relevant relationships that might exist between pairs of parameters (the full set of potentially relevant relations is in our replication package). In this context, we checked those relations by analyzing the contingency table [3] of each potentially relevant pair of parameters. The results of this analysis are described below.

8.1 Execution strategy versus UML diagrams

As discussed in Sect. 5.1, the most commonly required UML diagrams for model execution are class diagram, activity diagram, and state machine diagram, often in combination. Nevertheless, it is interesting to understand which diagrams

[19] http://projects.eclipse.org/projects/modeling.mdt.papyrus/who.

Table 6 Applied type of limitations and unsolved challenges

Type of limitations and unsolved challenges	#Studies	Studies
Expressiveness enhancement (e.g., support for more UML constructs, more expressive action language for behavioral specifications)	31	T5, T16, P1, P4, P5, P6, P7, P10, P12, P13, P14, P20, P24, P26, P27, P29, P30, P31, P36, P37, P38, P46, P49, P51, P53, P56, P57, P59, P61, P62, P63
Tool enhancement (e.g., refinement of the modeling editor, better testing of prototype tools)	20	P1, P2, P4, P6, P7, P8, P9, P10, P13, P17, P22, P27, P36, P37, P40, P45, P46, P54, P55, P60
Additional analysis of models (e.g., support for other timing and scheduling analyses on the UML models, new analysis for checking the conformance of the UML models to specific requirements)	12	T5, T14, T1, P9, P10, P16, P21, P24, P35, P38, P41, P42
Additional evaluation (e.g., more extensive experimentation for evaluating the approach, test the approach on additional systems and/or application domains)	11	P2, P5, P14, P16, P31, P32, P40, P45, P47, P53, P58
Execution platform improvement (e.g., more efficient scheduling of tasks at runtime, improvement of generated Java code)	7	P2, P4, P5, P8, P47, P51, P63
Generated code optimization (e.g., generate more efficient C++ code, generate Java code which implements Android development best practices)	6	P3, P5, P6, P8, P28, P59
Execution correctness assessment (e.g., automated consistency checks between the models and generated code, test the simulated models to check if they behave as expected)	4	P3, P18, P23, P26
Traceability enhancement (e.g., link error messages shown at runtime with entities in the UML models, trace and visualize the execution steps in the simulation directly into the UML models)	4	P36, P39, P40, P57
Support for model checking (e.g., support for deadlock detection, reachability analysis via model checking)	4	T15, P25, P33, P49
Reusability enhancement (e.g., improve the UML models repository, ability to reuse model elements across projects)	3	P10, P21, P41
Platform-specific limitations (e.g., the UML class diagram is bound to the Java single-inheritance constraint, only soft deadlines supported because the code generation does not target a real-time OS)	2	P19, P23
Support for runtime models update (e.g., process definition modified at runtime and without restarting the process execution, modification of execution contracts at runtime)	2	P32, P33
Additional execution targets (e.g., generated code beyond the already supported languages, like C++ code, generation of executable BPMN specifications)	2	P3, P54
Platform independence enhancement (e.g., development of a generic model of computation, independence with respect to the targeted programming language when performing translation)	2	P26, P27
Execution strategies combination (e.g., combination of interpretation and dynamic compilation)	1	P34
Scalability (e.g., avoid state space explosion when interpreting the models at runtime)	1	P31
Portability enhancement (e.g., porting of the UML virtual machine into other non-x86- based platforms)	1	P55

are actually translated, interpreted, or compiled. Our horizontal analysis unveiled the following interesting facts. When considering *translational* approaches, the most required UML diagrams are state machine (46/70), class (41/70), and activity (25/70), whereas the least required ones are interaction diagram (2/70), timing diagram (1/70), and ad hoc diagram from a UML profile (1/70). We also report that there are UML diagrams which are predominantly used in translational approaches and almost never used in *interpretive* approaches, they are: use case (21 against 1), structure (18 against 1, since Moka (T14) provides executable structural modeling with the Precise Semantics of UML Composite Structures), sequence (18 against 1), component (17 against 1), deployment (11 against 0), and communication diagram (6 against 1). Finally, state machine diagrams are (3/3) are required in all approaches for translation to object code, fol-

lowed by activity diagrams (2/3); all the other diagrams are either required in one case or not required at all.

8.2 Application domain versus required UML profiles

From the results of our horizontal analysis, we discovered that most of the research studies proposing a cross-domain solution (21/24) did not use any explicit UML profile. Of those that proposed domain-specific solutions, most relied on ad hoc profiles, while a good portion relied on the standard MARTE profile (13/39).

8.3 Execution strategy versus production system

As anticipated, while the majority of translational approaches (50/70) provides for the execution of the generated code on the ultimate target platform, no interpretive approach does so. This suggests that interpretation approaches are currently primarily used for model-level simulation and model-level analyses, but not for actual implementation.

8.4 Execution strategy versus readiness level

While most interpretive approaches, except for the Cameo Simulation toolkit (T3), displayed a *LOW* readiness level, a significant number of translational approaches (26/70) exhibited higher readiness levels (either *MEDIUM* or *HIGH*). This may be explained by the fact that translational execution has historically been the preferred approach, partly because it allowed direct reuse of existing programming language technologies (e.g., compilers) and partly because early versions of UML were not precise or detailed enough to produce executable code. With UML2 and the introduction of fUML and Alf, UML has become a full-fledged implementation-capable language, which can be interpreted and directly translated to object code. But, it will take time for these technologies to reach the necessary maturity level.

8.5 Execution strategy versus publication year

Not surprisingly, the introduction of fUML triggered efforts based on interpretive solutions. In fact, most of the approaches providing interpretation (12/14) were published after the first formalization of fUML in February 2011. However, efforts toward direct translation to object code are lagging. (Two-third of them were realized before the introduction of fUML.) Not even the introduction of Alf (2013) has triggered tangible additional efforts in this direction; however, some initiatives (e.g., [9]) toward these goals have recently started.

8.6 Readiness level versus primary study type

Among the studies with *HIGH* readiness level, only two were research studies, while the remaining ones (12/14) were tools. This is due to the fact that, for a tool to reach a *HIGH* readiness level, requires significant investment in making it dependable and usable, which are typically not a primary concern of researchers in the domain.

8.7 Intended benefits versus publication year

There appears to be an interesting correlation between intended benefits and publication year of the analyzed primary studies. In fact, the majority of the solutions aiming at improving functional correctness of the modeled system through execution were published after the first formalization of fUML (2011). This seems to confirm the hypothesis by which fUML has provided UML with the means for modeling software systems with precise execution semantics, whose correctness can be automatically assessed.

8.8 Intended benefits versus execution strategy

Slightly more than half of the translational approaches (29/56) sought to reduce the effort needed for producing an executable artifact. On the other hand, the majority of interpretive solutions (12/14) focused on improving the functional correctness of the modeled software system. This supports the view that translation of models to programming languages is preferred when generating production artifacts, whereas interpretation is mostly used for validating and improving functional correctness earlier in the development cycle.

8.9 Execution strategy versus fUML standard compliance

As of this writing, fUML has yet to become the anticipated widely adopted reference for UML semantics for both translational and interpretive approaches. While in translational solutions the adoption of fUML is increasing but still lagging behind (7/36 after 2011), in interpretive solutions the trend is much more promising (6/10 after 2011).

8.10 Simulation versus fUML standard compliance

Perhaps surprisingly, among the solutions that provide simulation, the majority (40/49) are not based on the fUML standard. In fact, only a few solutions (3/14) out of those published after the introduction of fUML are actually based on fUML semantics.

8.11 Simulation versus model-level debugging

An interesting correlation seems to exist between simulation and model-level debugging. From the horizontal analysis, it was determined that the majority of solutions providing model-level debugging also provide a means for simulation of UML models. This could be because, in most cases, simulation systems tend to incorporate mechanisms for model-level debugging to assist in determining functional correctness. Rightly or wrongly, such mechanisms are often perceived as excessive overhead in production-based implementations.

8.12 Covered MDA layers versus execution strategy

We can notice that the vast majority of approaches (68/82) consider UML models representing platform-independent aspects of the system and apply a translational model execution strategy. This is not surprising since those two categories are the most frequent ones in their respective parameters (see vertical analysis). Platform- and hardware-specific aspects are most often modeled when dealing with the translational execution strategy (36/70 and 16/70, respectively).

The interpretive execution strategy is in most cases applied when considering models representing platform-independent aspects of the system (14/14). We expected this result since model interpretation allows designers to achieve a high level of flexibility, which is more naturally achieved when abstract, platform-independent aspects are considered (rather than, e.g., hardware-specific ones). Nevertheless, given the continuously growing trend of cloud- and container-based solutions [15,16], we may expect that, in the future, interpretation of UML models will consider also platform- and hardware-related aspects of the system (e.g., for allowing the system to adapt its structure and behavior at runtime depending on the currently available infrastructural resources).

8.13 Identified limitations versus execution strategy

Interestingly, there are two main limitations when considering solutions based on UML interpretation, they are: better coverage of UML in terms of supported concepts (3/14) and support for model checking (3/14). These results can be seen as an indication about the need to (i) better support UML in terms of considered concepts and (ii) complement UML model interpretation with rigorous analysis in order to guarantee certain properties at runtime (e.g., safety, reliability, correctness). No relevant trends have been identified when considering solutions based on UML models translation.

8.14 Identified limitations versus support for non-functional properties

As discussed in Sect. 5.7, the majority of the solutions supporting non-functional properties focus on system performance (21/25). Among them we can notice three main identified limitations: expressiveness enhancement (13/25), tool enhancement (12/25), and additional analysis of models (5/25). All the other supported non-functional properties do not exhibit any specific trend.

8.15 Identified limitations versus support for simulation

When considering the support of simulation of UML models we noticed a certain balance with respect to solutions supporting simulation and those that do not support it. However, for the limitation related to the support of additional analysis of models this balance is not there anymore. Indeed, solutions with simulation have the limitation of providing additional analysis of the models in eight cases, against only one case for solutions without simulation. This result may be an indication of the need for complementing UML models simulation with additional analyses, along with the actual model execution.

8.16 Identified limitations versus readiness level

The data about solutions with *LOW* readiness level do not show any interesting trend with respect to their identified limitations, probably because it is the most recurrent readiness level (56/82). On the other hand, solutions with *MEDIUM* readiness level have three recurrent limitations: tool enhancement (5/12), support for additional analysis of models (4/12), and expressiveness enhancement (4/12). Finally, among the solutions with *HIGH* readiness level, only two identified some limitations (i.e., additional analysis of models, expressiveness enhancement, additional targets, code optimization, and execution correctness), all the other studies do not discuss any limitation of their proposed solution.

8.17 Production system versus support for simulation

Among the primary studies supporting simulation (49/82), there is a certain balance between whether the UML models are also executed in the production system (23/49) or not (26/49). This may be seen as an indication of the fact that simulation is deemed useful in both cases. The results are more unbalanced when considering primary studies not supporting simulation (33/82), where all of them support also the execution in production systems. This result is quite expected

since if an approach does not support simulation, then it is very likely that the models are executed in production.

8.18 Production system versus software platform

According to our vertical analysis, a total of 53/82 primary studies are able to execute models in production (e.g., full code generation and execution). Among the approaches supporting the execution in production, more than half of them do not prescribe a specific software platform for executing the generated code (31/53 vs. 19/53–3/53 no information available), meaning that in those cases the generated code is executable without any additional software such as libraries, dedicated virtual machines. Conversely, when the models are not executable in production (29/82), the majority of them requires a software platform for being executed (19/29) and they do not only in one case (i.e., P27); in 9 studies the authors did not provide enough information about their need of a software platform for executing the models.

8.19 Support for simulation versus software platform

Among the primary studies supporting simulation (49/82), the majority of them (29/49 vs. 11/49–9/49 no information available) generate code meant to run on a specific software platform (e.g., in P8 the generated Java code is used as a backend for model simulation and communicates to the modeling environment - Eclipse Papyrus and Moka—via a dedicated connection layer). The 11 remaining studies where simulation is performed, but the generated code does not run on a specific platform mean that the generated code is running stand-alone (e.g., in P41 the SystemC code is fully generated and used for simulation purposes). When considering the primary studies not providing simulation (33/49), in the majority of them (21/33) the generated code does not run on a specific platform, whereas in 9 cases it does. (For the remaining 3/33 cases, there was no information about whether the approach involves a dedicated software platform.)

9 Discussion and future prospects

Based on the lessons learned in the course of this study, in this section we give our interpretation of the obtained findings by discussing (i) future prospects for the technical advancement of UML models execution (Sect. 9.1) and (ii) our views on potentially relevant research directions (Sect. 9.2).

9.1 Future prospects for technical advancement

At the time of this writing, it is evident that there has been significant investment by both industry and academia in UML and related methods and technologies. Consequently, as happened with some of the earliest programming languages, such as Fortran and Cobol, it appears likely that UML will retain its relevance in engineering practice for many years to come, long after more modern and more advanced modeling languages appear. With this in mind, the following list captures our projections of what might be some of the dominant development trends in the execution of UML models. Needless to say, even the most carefully considered predictions often turn out to be wrong,[20] but regardless of how the future unfolds, the results of our study tell us that these are potential future developments that *should* be given priority both by researchers and as tool developers:

- ability to execute abstract (high-level) and incomplete models;
- enhanced observability of executing models;
- enhanced control of model execution;
- directly compiled model executables;
- support for UML-compliant action languages;
- support for executing models based on UML profiles;
- integration of UML simulators into heterogeneous (multi-paradigm) simulation systems;

Each of these capabilities is discussed in detail below.

9.1.1 Ability to execute abstract (high-level) and incomplete models

Throughout the entire history of engineering, models have been used as primary tools in support of both analysis and design activities. In analysis, they help designers to develop the necessary understanding of the problem and, more generally, the problem domain. In that context, an executable model can help validate the accuracy of that understanding by demonstrating that the behavior of the executing model matches reality. For design, on the other hand, an executable model serves to determine the degree of adequacy of a proposed design approach. Because design space of complex software systems can often be very large (potentially infinite), for system architects it would be particularly useful if executable models were available very early in the design process, when most key design decisions are being made. Effective design space exploration requires the ability to rapidly and easily evaluate multiple design alternatives including the ability to quickly identify and discard ones that are not promising, thereby freeing up time and resources for greater focus on those that are. This, in turn, implies a very lightweight modeling process at high levels of abstraction. Such an approach not only allows a greater number of

[20] To quote the inimitable Yogi Berra: "It's tough to make predictions, especially about the future".

alternatives to be considered and evaluated within a given time interval, but it also reduces the effort invested in model construction. The latter is important because, from a psychological viewpoint, it is much easier to discard unpromising alternatives that did not require much effort to produce and evaluate than ones that demanded significant time and effort.

At first glance perhaps, the idea of high-level (i.e., abstract) models may appear to exclude the use of model execution, which, after all, requires sufficient detail to be specified to make it executable. However, modeling tools such as ObjecTime Developer,[21] developed in the early 1990s, demonstrated the feasibility and practicality of executing abstract and incomplete models. Such models typically have a minimum of their core behavior specified, only what is sufficient to make them partially executable. As might be expected, this requires specialized and sophisticated runtime support, that allows handling of ambiguous and incompletely specified situations that can occur during execution. For example, choosing which execution path to follow or providing necessary detailed information required to steer the execution can be done by transferring control to a human operator, who can provide the necessary input. As stated in Sect. 5.1, only one of the analyzed approaches provides execution of partial models.

9.1.2 Enhanced observability of executing models

Since one of the primary purposes of modeling languages is to help humans understand complex problems and to understand and predict the behaviors of proposed solutions, any advances in this direction are likely to be crucial over the next few years. Perhaps the most significant of these is *enhanced observability*. As systems become increasingly more complex, there is a correspondingly greater need to understand how these systems work and to determine what they are doing in a given situation or at a point in time. This includes both *online* observability—the viewing of a system's state and operation in real-time—as well as *off-line*, which is the ability to record, play back, and analyze logs or execution traces of a system operation either in the field or in a simulator. From the results of our study, we were not able to extrapolate a clear research focus on enhancing observability of executing models; given the importance of observability in code-based software development, if models are to replace code, powerful observability for models is needed too.

Clearly, if facilitating understanding is the motivating requirement here, then the human consumers of this capability must be able to view and interpret the observed information in the form that most directly reflects their concerns. In other words, what is needed is *viewpoint-driven observability*. Clearly, the technology needed to support this is going to be sophisticated—particularly in the online case—involving inverse transformations of runtime data into desired concern-specific formats. Combined with this, more sophisticated analysis tools will be needed to scan execution trace information to detect and flag specific phenomena of interest. For example, it may be necessary to detect the conjunction (either temporal or logical) between two or more events or states that occur in physically dispersed parts of the system.

In this context, it is important to stress the fact that executed models shall be observed only when they are in a consistent state (e.g., only before or after an atomic execution step), and when they have observable execution states that conform to the observations metamodel. Indeed, if the model being executed is observed while the execution engine is fully reorganizing parts of its managed data (or the model itself), then what is observed would not make sense from the point of view of the modeler. Likewise, this reflection is important also when considering control, as it is likely that the model will be able to receive stimuli from its (production or simulation) environment only when such consistent execution states are reached.

Another very useful capability of this type is the "angiotracing" method devised by Hrischuk et al. [23], which is a facility to identify causally connected event chains in a running system. Such sophisticated functionality requires complex instrumentation to be developed and integrated into the executing system. Furthermore, in the online scenario, the mechanisms involved must have very low overhead and reporting latency. A related concern here is the ability to dynamically select and "instrument" which elements of a running system are to be observed. This includes issues of scalability. For example, a "smart city" network might involve literally tens and maybe even hundreds of thousands of sensors, and it must be possible to easily select, with low overhead, which particular ones (and when) are to be observed.

9.1.3 Enhanced control of model execution

As a natural complement to execution observability is *execution control* of a running system. If models replace code, powerful methods for controlling their execution are needed, and the results of this study could not identify a strong research effort in this direction. This can be decomposed into two distinct capabilities:

1. The ability to start, stop, step, or reset the execution of a running system or, what is typically even more useful, to do so selectively, on individual parts of a running system. In certain environments, it may also be of great utility to control the rate of execution of a system; that is, to have the ability to slow down the execution rate

[21] https://en.wikipedia.org/wiki/ObjecTime_Developer.

to a pace that is more amenable to direct human observation;[22]
2. The ability to "steer" or redirect the execution of a system through selective human or machine intervention. This may involve the controlled injection or suppression of key events, so that, by such means, the system can be forced into a desired state.

The kinds of observability and controllability discussed above are naturally desirable in a simulation environment. However, we have already noted that online observability is also highly useful, particularly in situations where the behavior of a system diverges from what is considered acceptable or safe norms. It is highly likely, therefore, that there will be a strong push to provide such capabilities "in the field", that is, in deployed systems. This seems particularly relevant to the coming generation of "smart" systems of systems, which will often require fully available non-stop "24/7" operation. Such systems will evolve dynamically in place, which means that the traditional hard distinction between development and field environments will be blurred.

9.1.4 Directly compiled model executables

As noted, the dominant approach to generating executable code from models is currently based on a multi-step translational approach: the model is typically first translated into a program in some widely used third-generation programming language. This is then translated into corresponding binary computer code in the conventional manner, using the tools of the target programming language. Unfortunately, this approach has two serious drawbacks:

- The translation tools, such as compilers, that are used in the second step are carefully designed to take full advantage of the semantics of their language. This allows them to implement numerous semantics-specific optimizations to realize time and memory savings. However, since they are designed for their programming language, they are *unable to take advantage of UML-specific semantics*. Invariably, the result is suboptimal code;
- The intermediate step whereby the model is translated into a programming language equivalent creates an additional barrier to the link between the executable code and its source model. This is exacerbated by the fact that the programming language translator is insensitive to UML-specific semantics. (For example, it may optimize away some aspect that is critical for the inverse mapping.) Needless to say, this complicates any model observability strategies discussed above.

On reflection, we can see that the multi-step model translation process is quite analogous to the early and now outdated Cfront approach to compiling C++ programs. Cfront was ultimately rejected in favor of direct compilation precisely because it resulted in suboptimal and often excessively complex executable code [37]. Consequently, we expect that, once an adequate level of language and tool maturity is reached, we will see the emergence of UML model compilers (i.e., the aforementioned compilation approach), which translate models directly into executable code and which, as a result, can generate better optimized code. Some early research in this direction has already shown promise [8], and other research efforts supported by industry have recently started [9].

9.1.5 Support for UML-compliant action Languages

As noted, most current systems capable of executing UML models rely on a third-generation programming language, such as C++ or Java, to specify detailed action code. However, with the adoption of the fUML along with its corresponding concrete syntax Alf, it is possible to use these standard UML-compliant languages instead. The advantage is obvious: the semantics of fUML and Alf are based on and fully compliant with UML itself. This precludes the ever-present and not unlikely danger that action code specified using a different language can violate standard UML semantics. Consequently, we hope and expect to see increasing support for such UML-compliant action languages.[23] This potentially has an additional and important benefit: the possibility of supporting the execution of models based on UML profiles, as explained below.

9.1.6 Support for executing UML models based on UML profiles

Extensive practice with UML has shown that it is often used as the base for domain-specific modeling languages through its profile mechanism (e.g., SysML). Most such languages involve domain-specific interpretations (specializations) of corresponding base UML concepts. In the ideal case, it should be possible to "plug-in" such semantics into standard UML model execution engines in order to execute models specified in those domain-specific languages. "Injecting" domain-specific semantics into an existing UML execution engine in this way does not seem as far-fetched as it sounds, when one considers that the specialized semantics of such a language can be formally expressed in a standardized way, using the Alf language and the underlying execution framework defined within fUML (in fact, this approach was used

[22] This requires the ability to control the progress of time, something which can only be realized in simulation environments.

[23] Alf is only one of potentially many possible concrete action languages that could be defined using fUML.

to define the semantics of fUML itself as well as other major sections of standard UML). At the time of this writing, at least one tool, Papyrus, already supports this type of capability. The ability of an execution environment to adopt language-specific semantics is part of a more general need: the capacity to be customized to suit specific needs and situations.

9.1.7 Integration of UML simulators into heterogeneous (multi-paradigm) Simulation Systems

With software being used increasingly to control various physical processes (so-called cyber-physical systems [32]), the ability to readily and correctly integrate and synchronize the execution of UML model simulators into complex multi-paradigm (e.g., hybrid) simulation frameworks, involving components that simulate different kinds of components has become crucial. And, in line with the previously noted gradual elimination of the boundary between design time and deployment time execution, an enhanced ability to combine UML simulators and/or execution systems with actual running systems is paramount.

9.2 Research perspectives for UML model execution

As a by-product of our in-depth analysis, we also reached certain conclusions about potential research directions for executable UML models. While these are inherently subjective and speculative in nature, we feel that it is worthwhile noting them in the expectation that they might prove useful to researchers in this domain.

A potentially significant need we observed pertains to what is referred to here and elsewhere as "model-level debugging." As it can be seen from Fig 12g, only a few solutions provide support for this. Yet, what we are dealing with here is the general ability to observe and understand the behavior of an executing systems, whether it be in development or in actual field operation. Recall that a central tenet of MDE is to raise the level of abstraction of software specification such that it is closer to the problem domain. Why should this be limited just to the development environment? Given the increasing reliance of modern society on software-intensive systems, the ability to observe, understand, and control executing software is gaining in significance. (Debugging, albeit critically important, is just one specialized manifestation of this more general but under-appreciated need.)

In concrete terms, this requirement translates into the general ability to observe and control executing models at an abstraction level that exactly matches the level in which they were specified. From a practical viewpoint, this can be decomposed into two finer-grained requirements: (1) the ability to select what is to be observed in an executing model and when, and (2) the ability to direct both the flow and even the pace of execution. This requires refined mechanisms to attach probes at selected points in time and space, to selectively capture and record the flow and consequences of execution (or, possibly, to slow it down to "human" rates), as well as to steer execution in the desired direction by selective injection of inputs or setting of variable values. All of this has to be done at the "model level," or the benefits of raising the level of abstraction will not only be lost, but may actually prove to be an impediment because of the need to understand the potentially complex relationship between a model and its low-level technology-specific realization. Clearly, this is a critical avenue of research for those working on model execution.

Note that the ability to control model execution is also critical when executable models are used during design space exploration. In those situations, it is desirable to produce minimal models that capture only the essence of the design approach, without investing excessive effort in specifying low-level implementation detail. Consequently, such models are inherently incomplete. This means that in the course of executing such a model, points can be reached where execution cannot proceed due to missing information or ambiguity. In such situation, the ability to "force" the model to proceed down a selected execution path becomes fundamental.

To summarize, we see that there is a critical need to research and develop new and efficient methods for observing and controlling the execution of software systems developed using MDE principles, including those based on executable UML.

10 Conclusions

In this study, we reported on a systematic review focused on identifying, classifying, and evaluating existing solutions, both in research and industry, for the execution of models based on the UML family of languages. The goal was to identify and assess trends, technical characteristics, available evidence, and limitations of current solutions for the benefit of both researchers and practitioners, as well as to identify a set of critical research challenges.

From over 5400 entries dealing with UML model execution, we selected 63 representative research studies and 19 tools. Research studies were selected via automatic searches on electronic data sources and a closed recursive backward and forward snowballing activity, whereas tools were selected via automatic searches on generic web search engines, from research studies, and by consultations with experts. Next, we derived a classification framework to help us characterize the different solutions, which we then applied to the 82 selected solutions. Finally, we analyzed and discussed the obtained data to: (i) provide an overview of the state of the art and practice in the domain and (ii) identify emergent research challenges and associated implications.

From the collected data we can conclude that: (i) there is growing scientific interest in UML model execution; (ii) currently, translational execution approaches clearly outnumber interpretive methods; (iii) there is a lack of solutions available for executing high-level (i.e., partial or incomplete) UML models; (iv) there are very few solutions available for model-level debugging; (v) very few solutions explicitly provide mechanisms for extension and customization; (vi) there is insufficient research of industrial usage and associated empirical methods; (vii) although most of the selected research studies were of good technical quality, many of them fail to adequately describe the context in which the research was performed, the roles of the researchers, and the limitations of their contributions; (viii) the limited degree of coverage of UML concepts (and, hence, the expressiveness) is the most common limitation of the analyzed solutions.

Acknowledgements This research was supported by the Knowledge Foundation through the SMARTCore (http://www.es.mdh.se/projects/377-SMARTCore) and MOMENTUM projects (http://www.es.mdh.se/projects/458-MOMENTUM).

Appendix A—Research team

Three researchers carried out this study, each of them with a specific role within the research team.

Principal researcher Dr. Federico Ciccozzi, senior researcher with expertise in model-driven engineering and component-based software engineering for the development of complex systems based on domain-specific modeling languages, both UML and EMF based. This team member specializes in: definition of DSMLs (including UML profiles), automatic model manipulations through transformations for code generation, analysis, model optimization, system properties preservation (just to mention a few); he was involved in all the activities, i.e., planning the study, conducting it, and reporting.

Research methodologist Dr. Ivano Malavolta, senior researcher with expertise in empirical methods applied to software systems and systematic literature reviews; he was mainly involved in (i) the planning phase of the study, and (ii) supporting the principal researcher during the whole study, e.g., by reviewing the data extraction form, the selected primary studies, the extracted data, the produced reports.

Advisor Prof. Bran Selic, senior researcher with extensive expertise in model-driven engineering, complex systems, UML and related profiles, simulation and execution of UML models. He provided guidance on key decisions and during conflicts, thereby avoiding 'endless discussions' [52]. He also supported the other researchers during data and findings synthesis activities.

From a geographical point of view, the research team is entirely distributed thereby ensuring 'expertise' and reinforcing 'independence' of the individuals' reviews.

Appendix B—Selected primary studies

See Tables 7 and 8.

Table 7 List of primary research studies

Study	Title	Author	Venue	Year
P1	Model-level, Platform-independent Debugging in the Context of the Model-driven Development of Real-time Systems	Bagherzadeh, M. et al.	Joint Meeting on Foundations of Software Engineering	2016
P2	Integration of UML models in FMI-Based co-simulation	Guermazi, S. et al.	Symposium on Theory of Modeling and Simulation	2016
P3	On the automated translational execution of the action language for foundational UML	Ciccozzi, F.	Software and Systems Modeling	2016
P4	A Model-Driven Engineering Methodology to Design Parallel and Distributed Embedded Systems	Enrici, A. et al.	ACM Transactions on Design Automation of Electronic Systems	2016
P5	System-level design based on UML/MARTE for FPGA-based embedded real-time systems	Leite, M. et al.	Design Automation for Embedded Systems	2016
P6	On the Generation of Full-Fledged Code from UML Profiles and ALF for Complex Systems	Ciccozzi, F. et al.	International Conference on Information Technology-New Generations	2015
P7	Animated simulation of integrated UML behavioral models based on graph transformation	Ermel, C. et al.	IEEE Symposium on Visual Languages and Human-Centric Computing	2015
P8	UML model execution via code generation	Dvai, G. et al.	International Workshop on Executable Modeling	2015
P9	UmpleRun: A dynamic analysis tool for textually modeled state machines using umple	Aljamaan, H. et al.	International Workshop on Executable Modeling	2015
P10	A customizable execution engine for models of embedded systems	Zurowska, K. et al.	Behavior Modeling—Foundations and Applications	2015
P11	HDL code generation from UML/MARTE sequence diagrams for verification and synthesis	Ebeid, E. et al.	Design Automation for Embedded Systems	2015
P12	Model-Driven Design of Network Aspects of Distributed Embedded Systems	Ebeid, E. et al.	IEEE Transactions on Integrated Circuits and Systems	2015
P13	Formal verification and validation of embedded systems: the UML-based MADES approach	Baresi, L. et al.	Software and Systems Modeling	2015
P14	A formal, model-driven design flow for system simulation and multi-core implementation	Diallo, P.I. et al.	IEEE International Symposium on Industrial Embedded Systems	2015
P15	A Model-Driven Approach to Generate Mobile Applications for Multiple Platforms	Usman, M. et al.	Asia-Pacific Software Engineering Conference	2014
P16	Improving the design flow for parallel and heterogeneous architectures running real-time applications: The PHARAON FP7 project	Posadas, H. et al.	Microprocessor and Microsystems	2014
P17	Reliable execution of statechart-generated correct embedded software under soft errors	Ferreira, R.R. et al.	Design and Diagnostics of Electronic Circuits and International Symposium on Systems	2014
P18	Formalizing execution semantics of UML profiles with fUML models	Tatibouet, J. et al.	Model-Driven Engineering Languages and Systems	2014
P19	Textual, executable, translatable UML	Dévai, G. et al.	OCL@ MoDELS	2014
P20	Extending UML/MARTE to Support Discrete Controller Synthesis, Application to Reconfigurable Systems-on-Chip Modeling	Guillet, S. et al.	ACM Transactions on Reconfigurable Technology and Systems	2014
P21	Model-driven engineering of Manufacturing Automation Software Projects A SysML-based approach	Vogel-Heuser, B. et al.	Mechatronics	2014
P22	A model-based framework for developing real-time safety Ada systems	Salazar, E. et al.	Reliable Software Technologies–Ada-Europe	2013
P23	Environment modeling and simulation for automated testing of soft real-time embedded software	Iqbal, M.Z. et al.	Software and Systems Modeling	2013
P24	Combining fUML and profiles for non-functional analysis based on model execution traces	Berardinelli, L. et al.	International ACM Sigsoft conference on Quality of software architectures	2013
P25	Executing and debugging UML models: an fUML extension	Laurent, Y. et al.	ACM Symposium on Applied Computing	2013
P26	Multi-Paradigm Semantics for Simulating SysML Models using SystemC-AMS	Chaves Cafe, D. et al.	Specification and Forum on Design Languages	2013

Table 7 continued

Study	Title	Author	Venue	Year
P27	A Plug-in Based Approach for UML Model Simulation	Radjenovic, A. et al.	European conference on Modelling Foundations and Applications	2012
P28	A Model Driven Approach for Android Applications Development	Parada, A.G. et al.	Brazilian Symposium on Computing System Engineering	2012
P29	Modeling and simulation of secure wireless sensor network	Diaz, A. et al.	Forum on Specification and Design Languages	2012
P30	An Optimized Compilation of UML State Machines	Charfi, A. et al.	International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing	2012
P31	Symbolic execution of UML-RT state machines	Zurowska, K. et al.	ACM Symposium on Applied Computing	2012
P32	Achieving process modeling and execution through the combination of aspect and model-driven engineering approaches	Bendraou, R. et al.	Journal of Software: Evolution and Process	2012
P33	Contracts for Model Execution Verification	Cariou, E. et al.	European conference on Modelling foundations and applications	2011
P34	On the Performance of UML State Machine Interpretation at Runtime	Höfig, E. et al.	International Symposium on Software Engineering for Adaptive and Self-Managing Systems	2011
P35	Framework to Simulate the Behavior of Embedded Real-Time Systems Specified in UML Models	Wehrmeister, M.A. et al.	Brazilian Symposium on Computing System Engineering	2011
P36	Modelica code generation from ModelicaML state machines extended by asynchronous communication	Pohlmann, U. et al.	International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools	2011
P37	Code generation for UML 2 activity diagrams: Toward a comprehensive model-driven development approach	Gessenharter, D. et al.	European conference on Modelling foundations and applications	2011
P38	From UML to Petri Nets: The PCM-Based Methodology	Distefano, S. et al.	IEEE Transactions on Software Engineering	2011
P39	Closing the Gap between UML-based Modeling, Simulation and Synthesis of Combined HW/SW Systems	Mischkalla, F. et al.	Design, Automation and Test in Europe	2010
P40	Matilda: A Generic and Customizable Framework for Direct Model Execution in Model-Driven Software Development	Wada, H. et al.	Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization	2009
P41	SystemC/C-Based Model-Driven Design for Embedded Systems	Riccobene, E. et al.	ACM Transactions on Embedded Computing Systems	2009
P42	Performance evaluation of UML2-modeled embedded streaming applications with system-level simulation	Arpinen, T. et al.	EURASIP Journal on Embedded Systems	2009
P43	A co-design approach for embedded system modeling and code generation with UML and MARTE	Vidal, J. et al.	Design, Automation and Test in Europe	2009
P44	SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications	Moebius, N. et al.	International Conference on Availability, Reliability and Security	2009
P45	Realization of UML class and state machine models in the C# code generation and execution framework	Derezinska, A. et al.	Informatica	2009
P46	eUDEVS: Executable UML with DEVS Theory of Modeling and Simulation	Risco-Martin, J.L. et al.	Simulation	2009
P47	Model-driven development of composite context-aware web applications	Kapitsaki, G.M. et al.	Information and Software Technology	2009
P48	Execution and Simulation of (Profiled) UML Models using Pópulo	Fuentes, L. et al.	International workshop on Models in software engineering	2008
P49	Toward a UML virtual machine: implementing an interpreter for UML 2 actions and activities	Crane M.L. et al.	Conference of the center for advanced studies on collaborative research: meeting of minds	2008
P50	Automatic Performance Model Transformation from UML to C++	Pllana, S. et al.	International Conference on Parallel Processing—Workshops	2008

Table 7 continued

Study	Title	Author	Venue	Year
P51	An Execution Framework for MARTE-based Models	Mraidha, C. et al.	International Conference on Engineering of Complex Computer Systems	2008
P52	UJECTOR: A Tool for Executable Code Generation from UML Models	Usman, M. et al.	Advanced Software Engineering and Its Applications	2008
P53	MDD4SOA: Model-Driven Service Orchestration	Mayer, P. et al.	International IEEE Enterprise Distributed Object Computing Conference	2008
P54	Enabling the Evolution of Service-Oriented Solutions Using an UML2 Profile and a Reference Petri Nets Execution Platform	Fabra, J. et al.	International Conference on Internet and Web Applications and Services	2008
P55	A Model-Based Approach for Platform-Independent Binary Components with Precise Timing and Fine-Grained Concurrency	Schattkowsky, T. et al.	Hawaii International Conference on System Sciences	2007
P56	FSMC+, a tool for the generation of Java code from statecharts	Tiella, R. et al.	International Symposium on Principles and Practice of Programming in Java	2007
P57	MDA-based approach for embedded software generation from a UML/MOF repository	Do Nascimento, F.A.M. et al.	Symposium on Integrated Circuits and Systems Design	2006
P58	A Model-Based Approach for Executable Specifications on Reconfigurable Hardware	Schattkowsky, T. et al.	Design, Automation and Test in Europe	2005
P59	Automatic Code Generation from a UML model to JEC 61131-3 and system configuration tools	Vogel-Heuser, B. et al.	International Conference on Control and Automation	2005
P60	Embedded System Design Using Formal Model Refinement: An Approach Based on the Combined Use of UML and the B Language	Voros, N.S. et al.	Design Automation for Embedded Systems	2004
P61	Deriving executable process descriptions from UML	Di Nitto, E. et al.	International Conference on Software Engineering	2002
P62	A UML tool for an automatic generation of simulation programs	Arief, L.B. et al.	International Workshop on Software and Performance	2000
P63	Testing and simulating production control systems using the Fujaba environment	Niere, J. et al.	Applications of Graph Transformations with Industrial Relevance	2000

Table 8 List of primary tools

Tool	Name	URL
T1	ARTISAN Studio Sysim	http://www.atego.com/products/sysim/
T2	BOUML	http://www.bouml.fr/
T3	Cameo Simulation toolkit	http://www.nomagic.com/products/magicdraw-addons/cameo-simulation-toolkit.html
T4	CHESS	https://www.polarsys.org/projects/polarsys.chess
T5	Fujaba	http://www.fujaba.de/
T6	IBM Rational Rhapsody (family)	http://www.ibm.com/developerworks/downloads/r/rhapsodydeveloper/
T7	IBM Rational Rose (family)	http://www-03.ibm.com/software/products/en/ratirosefami
T8	IBM Rational Software Architect	http://www-03.ibm.com/software/products/en/ratisoftarch
T9	IBM RSARTE	http://www-01.ibm.com/support/docview.wss?uid=swg27041556
T10	IBM Rational Tau	http://www-03.ibm.com/software/products/en/ratitau
T11	Abstract Solutions iUML	http://www.abstractsolutions.co.uk/PRODUCTS/iuml/
T12	One Fact BridgePoint	https://xtuml.org/
T13	fUML implementation	http://fuml.modeldriven.org
T14	Moka	https://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution
T15	Qompass	https://wiki.eclipse.org/Papyrus_Qompass
T16	Papyrus-RT	https://www.eclipse.org/papyrus-rt/
T17	Quantum Leaps QM	http://www.state-machine.com/qm/
T18	Sparx Enterprise Architect	http://www.sparxsystems.com/
T19	Syntony	http://www7content.cs.fau.de/syntony/

Appendix C—Extracted data for each primary study

Legend for Table 9. **UML diagrams** (UC: Use case, T: Timing, STRUCT: Comp. structure, SM: State machine, SEQ: Sequence, P: Package, O: Object, INT: Interaction, DEP: Deployment, C: Component, COMM: Communication, CL: Class, ACT: Activity, COLLAB: Collaboration, AH: Ad hoc). **Action lang.** (ADA: ADA, UAL: UAL, C#: C#, C: C, ALF: ALF, J: Java, C++: C++, UA: UML actions, ✗: Not supported). **Profiles** (MOD: ModelicaML, ✗: Not supported, PA: UML-PA, SPT: UML-SPT, SC: SystemC UML profile, RT: UML-RT, SYS: SysML, M: MARTE, AH: Ad hoc UML profile). **Tool** (I: Tool independent, C: CHESS, S: Sparx Enterprise Architect, RR: IBM Rational Rose, A: Artisan Real-Time Studio, RSA: IBM Rational Software Architect, EU: Eclipse UML2, P: Eclipse Papyrus). **fUML** (✓: yes, ✗: no). **MDA levels** (PIM: platform-independent model, PSM: platform-specific model, HW: hardware).

Legend for Table 10. **Execution strategy** (I: interpretation, T: translation). **Intended benefits** (C: correctness, P: production, Q: quality). **Associated process** (✓: yes, ✗: no). **Extensibility** (✓: yes, ✗: no). **Readiness level** (H: high, M: medium, L: low). **Non-functional properties** (C: code size, P: performance, A: adaptability, S: safety, SE: security, ✗: not supported). **Formal languages** (B: Event B, BHDL: BHDL, FFSM: Finite state machine (ad hoc), Jolie: jolie, KIV: KIV, Lisp: Lisp, NuSMV: nusmv, Z: Z, ✗: not supported).

Legend for Table 11. **Traceability** (✓: yes, ✗: no). **Model debugging** (✓: yes, ✗: no). **Simulation** (✓: yes, ✗: no). **Production system** (✓: yes, ✗: no).

Legend for Table 12. **Software platform** (✓: yes, ✗: no).

Legend for Table 13. **Research method** (V: Validation, E: Evaluation). **Type of evidence** (E: Example, ES: Set of experiments, EXS: Set of examples, EL: Empirical experiment in the lab, EXI: Example from industry, EXSI: Set of examples from industry, EI: Industrial empirical experiment, EVI: Industrial evaluation). **Type of systems for evidence** (W: Web, S: Synthetic, SE: Monitoring and sensing, MR: Mobile robotic system, A: Mobile app, MC: Media converter, MA: Mathematical function, MAN: Manufacturing, L: Modeling language, IS: Information system, C: Controller, CE: Consumer electronics, COM: Communication, B: Business processes). **Identified limitations** (T: Traceability enhancement, TO: Tool enhancement, S: Scalability, R: Support for runtime models update, RE: Reusability enhancement, P: Portability enhancement, PI: Platform independence enhancement, EP: Execution platform improvement, MC: Support for model checking, E: Expressiveness enhancement, EC: Execution strategies combination, ECO: Execution correctness assessment, O: Generated code optimization, COV: Better coverage of UML, EV: Additional evaluation, AN: Additional analysis of models, PS: Platform-specific limitations) (Table 14).

Table 9 Technical characteristics (RQ2)—UML modeling

Study	UML diagrams	Action lang.	Profiles	Tool	fUML	MDA levels
P01	C, SM	UA	RT	P	✗	PIM
P02	C, ACT, CL	UA	AH	P	✓	PIM
P03	CL	ALF, C++	✗	P	✓	PIM
P04	SEQ, ACT, P, DEP, C	UA	SYS, AH	Other	✗	PIM, PSM, HW
P05	CL, SEQ	UA	M	Other	✗	PIM, PSM
P06	C, STR, CL, DEP, SM	ALF	M, Other	C, P	✓	PIM, PSM, HW
P07	CL, O, SM, COMM, UC	UA	✗	Other	✗	PIM
P08	CL, SM	ALF	✗	P	✓	PIM
P09	CL, SM	UA	✗	Other	✗	PIM
P10	C, SM	UA	RT	P	✗	PIM
P11	SEQ	✗	M	P	✗	PIM, PSM
P12	DEP	✗	SC, Other	P, I	✗	PSM, HW
P13	CL, SEQ, SM	✗	M, AH	I	✗	PIM, PSM
P14	CL, STR, SM	C, C++, UA	AH	Other	✗	PIM, PSM, HW
P15	CL, SM	ALF	AH	EU	✗	PIM
P16	C, STR, DEP	✗	M	EU	✗	PIM, PSM, HW
P17	SM	UA	✗	Other	✗	PIM
P18	ACT	ALF	✗	P	✓	PIM
P19	SM	Other	✗	P	✗	PIM
P20	C, STR, SM	✗	M, Other	P	✗	PIM
P21	AH	✗	SYS, Other	Other	✗	PIM, PSM, HW
P22	C, STR, CL, SEQ	✗	M	RSA	✗	PIM, PSM
P23	CL, SM	J	M, AH	I	✗	PIM
P24	CL, ACT, O	UA	M	EU	✓	PIM, PSM, HW
P25	CL, ACT	UA	✗	EU, P	✓	PIM
P26	STR, SM	✗	SYS	EU	✗	PIM
P27	SM, CL	Other	✗	Other	✗	PIM
P28	CL, SEQ	✗	✗	I, P	✗	PIM
P29	C, STR	✗	M	–	✗	PIM, PSM
P30	SM, ACT	UA	✗	P	✓	PIM

Table 9 continued

Study	UML diagrams	Action lang.	Profiles	Tool	fUML	MDA levels
P31	SM	C++	RT	RSA	✗	PIM
P32	ACT, CL	UA	Other	EU	✗	PIM
P33	SM	✗	✗	EU	✗	PIM
P34	SM	✗	✗	EU	✗	PIM
P35	ACT, SM, SEQ	UA	M	I	✗	PIM
P36	SM, C, STR	Other	MOD	P	✗	PIM
P37	CL, ACT	UA	✗	EU	✗	PIM
P38	UC, O, ACT, DEP	UA	SPT	Other	✗	PIM, PSM, HW
P39	ACT, STR	C++	SYS, AH	A	✗	PIM, PSM, HW
P40	CL, SEQ	✗	Other	I	✗	PIM, PSM
P41	CL, C, STR, SM, O, DEP	C, C++, Other	SC, AH	S	✗	PIM, PSM, HW
P42	ACT, STR	UA	M, AH	I	✗	PIM, PSM, HW
P43	CL, C, STR, SM, O	C++	M, SYS	Other	✗	PIM, PSM, HW
P44	CL, ACT, DEP	UA	AH	Other	✗	PIM, PSM
P45	CL, SM	✗	✗	I	✗	PIM
P46	P, CL, C, STR, SEQ, SM, T	✗	✗	RSA	✗	PIM
P47	CL, ACT, SM	✗	AH	EU	✗	PIM
P48	CL, ACT	UA	✗	I	✗	PIM
P49	ACT	UA	✗	Other	✗	PIM
P50	ACT	C++	✗	Other	✗	PIM
P51	CL, SM	UA	M	Other	✗	PIM
P52	CL, SEQ, ACT	UA	✗	Other	✗	PIM
P53	ACT	UA	Other	I	✗	PIM, PSM
P54	SM, COLLAB	UA	AH	Other	✗	PIM
P55	ACT, SM, INT	UA	Other	Other	✗	PIM, PSM
P56	SM	J	✗	I	✗	PIM
P57	COMM, SEQ	✗	SPT	I	✗	PIM, PSM
P58	CL, SM, SEQ	UA	Other	Other	✗	PIM, PSM, HW
P59	CL, COMM, SEQ, DEP	✗	PA	A	✗	PIM, PSM, HW
P60	P, CL, SM	Other	AH	RR	✗	PIM, PSM, HW

Table 9 continued

Study	UML diagrams	Action lang.	Profiles	Tool	fUML	MDA levels
P61	ACT, CL, SM	✗	✗	I	✗	PIM
P62	CL, SEQ	✗	✗	Other	✗	PIM
P63	CL, ACT, COMM, SM	UA, J, Other	✗	Other	✗	PIM
T01	–	–	SYS	A	✗	–
T02	CL, SM, ACT, DEP	J, C++, Other	✗	Other	✗	PIM
T03	SM, ACT	Other	SYS	Other	✓	PIM
T04	CL, C, STR, SM, DEP, O	ALF	M, Other	C, P	✓	PIM, PSM, HW
T05	CL, ACT, COMM, SM	UA, J, Other	✗	Other	✗	PIM
T06	ACT, SM, CL	ALF, J, C++, C, C#, ADA	SYS	Other	✗	PIM, PSM
T07	CL	UAL, ADA, C++, Other, J	✗	RR	✗	PIM, PSM
T08	ACT, INT, SM, CL, SEQ	UAL, J, C++, C#	✗	RSA	✗	PIM, PSM
T09	ACT, SEQ, C, STR, CL, SM	UAL, J, C++, C	RT	Other	✗	PIM, PSM
T10	CL, ACT, SM	Other, J, C, C++, C#	✗	Other	✗	PIM, PSM
T11	CL, SM, UC, SEQ, COMM	Other	Other	Other	✗	PIM
T12	CL, SM	Other	Other	Other	✗	PIM, PSM
T13	CL, ACT	ALF	✗	I	✓	PIM
T14	CL, ACT, SM	ALF	✗	P	✓	PIM
T15	C, SM, STR, DEP	C++, Other	M, AH	P	✗	PIM, PSM, HW
T16	CL, C, STR, SM	C++	Other	P	✗	PIM, PSM
T17	SM, ACT	C, C++	✗	Other	✗	PIM, PSM
T18	CL, SM, SEQ, ACT	J, C, C++	✗	S	✓	PIM, PSM
T19	C, STR, SM, ACT	✗	AH	Other	✗	PIM, PSM

Table 10 Technical characteristics (RQ2)—Modeling execution solutions

Study	Execution strategy	Intended benefits	Associated process	Extensibility	Readiness level	Non-functional properties	Formal specification languages
P01	T	P	✗	✗	L	✗	✗
P02	I	C, P, Q	✗	✗	L	✗	✗
P03	T	P	✗	✗	H	✗	✗
P04	T	P, Q, C	✓	✗	L	P, A	Other
P05	T	P, Q	✓	✗	L	P, Other	✗
P06	T	C, Q, P	✓	✗	M	P	✗
P07	T	C	✗	✗	L	✗	✗
P08	T	C	✗	✗	L	✗	✗
P09	T	C	✗	✗	L	✗	✗
P10	T	C	✗	✓	L	✗	Other
P11	T	C, P	✗	✗	L	✗	✗
P12	T	Q	✗	✗	L	P	✗
P13	T	C, Q, P	✓	✗	M	P	Lisp
P14	T	P, C	✓	✗	L	✗	Other
P15	T	P	✗	✗	M	✗	✗
P16	T	C, Q, P	✗	✗	M	P	✗
P17	T	C	✗	✗	L	✗	NuSMV
P18	I	C	✗	✗	L	✗	✗
P19	T	C	✗	✗	L	✗	✗
P20	T	P	✗	✗	L	P	✗
P21	T	P	✓	✗	M	✗	✗
P22	T	Q, P	✗	✗	M	P	✗
P23	T	C	✗	✗	M	P	✗
P24	I	Q	✗	✗	L	P	✗
P25	I	C	✗	✗	L	✗	✗
P26	T	C	✗	✗	L	✗	✗
P27	T	C, Q	✗	✓	M	S	✗
P28	T	P	✗	✗	L	✗	✗
P29	T	Q	✗	✗	L	SE, P	✗
P30	T	P, Q	✗	✗	L	C	✗

Table 10 continued

Study	Execution strategy	Intended benefits	Associated process	Extensibility	Readiness level	Non-functional properties	Formal specification languages
P31	I	C	✗	✗	L	✗	Other
P32	I	C	✗	✗	L	✗	✗
P33	I	C	✗	✗	L	✗	✗
P34	I	P, Q	✗	✗	L	P, A	✗
P35	T	C	✗	✗	L	✗	✗
P36	T	C	✗	✗	L	✗	✗
P37	T	P	✗	✗	L	✗	✗
P38	T	Q	✗	✗	L	P	Other
P39	T	P	✗	✗	L	✗	✗
P40	T	Q	✗	✓	M	P	✗
P41	T	P, Q	✗	✗	M	P	✗
P42	T	Q	✗	✓	L	P	✗
P43	T	P	✗	✗	L	✗	✗
P44	T	Q, C	✗	✗	L	SE	KIV
P45	T	P, C	✗	✗	L	✗	✗
P46	T	C	✗	✗	L	✗	✗
P47	T	P	✗	✗	L	✗	✗
P48	I	C	✗	✓	L	✗	✗
P49	I	C	✗	✗	L	✗	✗
P50	T	Q	✗	✗	L	P	✗
P51	T	P	✗	✗	L	P	✗
P52	T	P	✗	✗	L	✗	✗
P53	T	P	✗	✗	L	✗	Jolie
P54	T	Other, P	✓	✗	L	✗	Other
P55	T	Q	✗	✗	L	P	✗
P56	T	P	✗	✗	H	✗	NuSMV
P57	T	Q, P	✗	✗	L	P	✗
P58	T	Q	✗	✗	L	P	✗
P59	T	P	✗	✗	L	✗	✗
P60	T	Q, P, C	✓	✗	L	P	B, BHDL

Table 10 continued

Study	Execution strategy	Intended benefits	Associated process	Extensibility	Readiness level	Non-functional properties	Formal specification languages
P61	T	P	✗	✗	L	✗	Z
P62	T	C, Q	✗	✗	L	P	✗
P63	T	P, C	✗	✓	L	✗	✗
T01	T	C	✗	✗	H	✗	✗
T02	T	P	✗	✓	L	✗	✗
T03	T	C	✗	✗	H	✗	✗
T04	T	C, P, Q	✓	✗	M	P	✗
T05	T	P, C	✗	✓	M	✗	✗
T06	T	P, C	✗	✓	H	✗	✗
T07	T	P	✗	✓	H	✗	✗
T08	T	P, C	✗	✓	H	✗	✗
T09	T	P, C	✗	✓	H	✗	✗
T10	T	P, C	✗	✓	H	✗	✗
T11	T, I	P, C	✓	✓	H	✗	✗
T12	T, I	P, C	✓	✓	H	✗	✗
T13	I	C	✗	✗	L	✗	✗
T14	I	C	✗	✓	L	✗	✗
T15	T	P	✗	✗	L	✗	✗
T16	T	P, Q, C	✗	✓	H	P, Other	✗
T17	T	P	✗	✗	H	✗	✗
T18	T	P, C	✗	✓	H	✗	✗
T19	T	C	✗	✗	L	✗	✗

Table 11 Technical characteristics (RQ2)—Modeling execution strategies

Study	Interpr. engine	Target platform	Traceability links	Execution tools	Model debugging	Simulation support	Production system
P01	–	–	✓	EclipseUML2, Other, C++	✓	✓	✗
P02	Ad hoc	–	–	Other, EclipseUML2	✗	✓	✗
P03	–	–	✗	Other, EclipseUML2	✗	✓	✓
P04	–	–	✗	Other	✗	✓	✓
P05	–	–	✗	–	✗	✗	✓
P06	–	–	✓	XPand, QVT	✗	✗	✓
P07	–	–	✗	Graph T., Genged	✗	✓	✗
P08	–	–	✗	Other, EclipseUML2	✓	✓	✗
P09	–	–	✓	Other	✗	✓	✗
P10	–	–	✗	ATL, Other	✗	✓	✗
P11	–	–	✗	HIF	✗	✗	✓
P12	–	–	✗	HIF, C++	✗	✓	✗
P13	–	–	✓	Other	✗	✓	✗
P14	–	–	✗	C++, XSLT	✗	✓	✓
P15	–	–	✗	Other	✗	✗	✓
P16	–	–	✗	EclipseUML2	✗	✓	✓
P17	–	–	✗	C	✗	✓	✓
P18	Moka	–	–	Other	✗	✓	✗
P19	–	–	✗	Other	✓	✓	✓
P20	–	–	✗	Other	✗	✗	✓
P21	–	–	✗	MOFM2T	✓	✗	✓
P22	–	–	✗	QVT, MTL, RSA	✗	✗	✓
P23	–	–	✗	MOFScript	✗	✓	✗
P24	fUML	–	–	fUML, Other	✗	✓	✗
P25	fUML	–	–	fUML, Other	✓	✓	✗
P26	–	–	✗	ATL, Acceleo	✗	✓	✓
P27	–	–	✗	Prolog, Cassandra	✗	✓	✗
P28	–	–	✗	GenCode	✗	✗	✓
P29	–	–	✗	Acceleo	✗	✓	✗
P30	–	Gimple	✗	Acceleo, GCC	✗	✗	✓

Table 11 continued

Study	Interpr. engine	Target platform	Traceability links	Execution tools	Model debugging	Simulation support	Production system
P31	RSA	–	–	FFSM	✗	✓	✗
P32	Kermeta	–	–	Kermeta	✗	✗	✓
P33	Mocas	–	–	Other, Mocas, EclipseUML2	✓	✓	✗
P34	Ad hoc	–	–	Other, EclipseUML2, MVFlex	✗	✗	✓
P35	–	–	✗	Other	✗	✓	✗
P36	–	–	✗	Acceleo, Dymola, C, Sanz	✗	✓	✗
P37	–	–	✗	Other, EclipseUML2	✓	✗	✓
P38	–	–	✗	Other	✗	✓	✗
P39	–	–	✗	Artisan, TDK, DSL, Shadowacs, Agility SC	✗	✓	✓
P40	–	–	✗	Other	✗	✗	✓
P41	–	–	✓	VB	✗	✓	✓
P42	–	–	✗	Other	✗	✓	✗
P43	–	–	✗	MDWorkbench	✗	✗	✓
P44	–	–	✗	QVT, XPand	✗	✗	✓
P45	–	–	✗	Other	✗	✗	✓
P46	–	–	✗	XML, Other, XSLT	✗	✓	✗
P47	–	–	✗	Other, EclipseUML2, Apache Velocity	✗	✗	✓
P48	Populo	–	–	Populo	✓	✓	✗
P49	ACTi	–	–	Other	✗	✓	✗
P50	–	–	✗	Performance Prophet	✗	✓	✗
P51	–	–	✗	Accord UML	✗	✗	✓
P52	–	–	✗	Other	✗	✗	✓
P53	–	–	✗	Other	✗	✗	✓
P54	–	–	✗	XML, Other, XSLT	✗	✓	✓
P55	–	UVM	✗	Upad	✗	✗	✓
P56	–	–	✗	FSMC+	✗	✗	✓
P57	–	–	✗	QVT	✗	✗	✓
P58	–	Ad hoc	✗	AEP VM, Other	✗	✗	✓
P59	–	–	✗	–	✗	✗	✓

Table 11 continued

Study	Interpr. engine	Target platform	Traceability links	Execution tools	Model debugging	Simulation support	Production system
P60	–	–	✗	Atelier B	✗	✓	✓
P61	–	–	✗	XSLT	✗	✗	✓
P62	–	–	✗	Other	✗	✓	✗
P63	–	–	✗	Other	✓	✓	✓
T01	–	–	✗	C##	✗	✓	✗
T02	–	–	✗	QT	✗	✗	✓
T03	–	–	✗	Cameo, Other	✓	✓	✗
T04	–	–	✓	XPand, QVT, Other	✗	✗	✓
T05	–	–	✓	Other	✓	✓	✓
T06	–	–	✓	Other	✓	✓	✓
T07	–	–	✓	Other	✓	✗	✓
T08	–	–	✓	Other	✓	✓	✓
T09	–	–	✓	Other	✓	✓	✓
T10	–	–	✓	Other	✓	✓	✓
T11	Other	–	✓	Other, ASL	✓	✓	✓
T12	Other	–	✗	Other, XML	✓	✓	✓
T13	fUML	–	–	Other, fUML	✗	✗	✓
T14	Moka	–	–	Other	✓	✓	✗
T15	–	–	✗	Other, Acceleo	✗	✗	✓
T16	–	–	✗	C++	✗	✗	✓
T17	–	–	✓	C++	✗	✗	✓
T18	–	–	✓	Other	✓	✓	✓
T19	–	–	✗	Other, Omnet++	✓	✓	✗

Table 12 Technical characteristics (RQ2)—translation-specific data

Study	Translation steps	Intermediate artifacts	Transformation targets	Platform
P01	2	UML model	Other	✓
P02	–	–	–	–
P03	1	Other	C++	✗
P04	2	DSL model	C, C++, Other	✗
P05	1	Other	Other	✗
P06	2	DSL model	C++	✗
P07	1	–	Other	✓
P08	1	Other	J	✓
P09	1	Other	J	✓
P10	1	Other	Other	✓
P11	1	Other	SystemC, VHDL	✓
P12	2	Other	SystemC	✓
P13	2	Java	Other	✓
P14	2	DSL model	C++, SystemC, XML, VHDL	✓
P15	1	Other	J, C##	✗
P16	3	XML file, Textual file	C, C++, Other	✓
P17	1	Other	C	✗
P18	–	–	–	–
P19	1	Other	J	✗
P20	3	Other, Java	C	✗
P21	1	Other	Other	✗
P22	2	Other, UML model	ADA, Other	✗
P23	1	Other	J	✓
P24	–	–	–	–
P25	–	–	–	–

Table 12 continued

Study	Translation steps	Intermediate artifacts	Transformation targets	Platform
P26	2	DSL model	SystemC, VHDL	✗
P27	2	Other	Other	✗
P28	1	Other	J	✗
P29	1	Other	XML	✓
P30	1	Other	Other	✗
P31	–	–	–	–
P32	–	–	–	–
P33	–	–	–	–
P34	–	–	–	–
P35	1	Other	Other	✓
P36	1	Other	Other	✓
P37	2	UML model	J	✗
P38	2	Other	Other	✓
P39	2	Other	VHDL	✗
P40	1	Other	J	✓
P41	1	Other	SystemC, C, C++	✗
P42	2	Other	SystemC, XML	✓
P43	1	Other	VHDL, Other	✗
P44	3	UML model	J	✗
P45	1	Other	Other	✓
P46	1	XML file, Java	Other	✓
P47	1	Other	J, XML	✗
P48	–	–	–	–
P49	–	–	–	–
P50	2	XML file	C++	✓

Table 12 continued

Study	Translation steps	Intermediate artifacts	Transformation targets	Platform
P51	1	Other	C++	✗
P52	1	Other	J	✗
P53	3	DSL model	BPEL, WSDL, J, Other	✗
P54	1	Other	Other	✓
P55	1	Other	Other	✓
P56	2	Other	J, Other, HTML	✓
P57	3	DSL model	J	✗
P58	1	Other	Other	✓
P59	1	Other	Other	✗
P60	2	Formal specification	C, C++, VHDL, SystemC	✗
P61	2	XML file	J	✓
P62	2	DSL model	J	✓
P63	1	Other	J	✗
T01	Other	–	.net	✓
T02	1	Other	C++, J, PHP, Python, Other	✗
T03	Other	–	Other, J	✓
T04	3	UML model, DSL model	ADA, C++	✗
T05	1	Other	J	✗
T06	Other	Other	J, C++, C, C##, ADA	✓
T07	Other	Other	ADA, C++, Other, J, VB	✓
T08	Other	Other	J, C++, C##	✓
T09	Other	Other	Other, J, C++, C	✓
T10	Other	Other	J, C, C++, C##	✓
T11	3	DSL model	C, C++, ADA, J	✓
T12	1	Other	C, C++, SystemC	✓
T13	–	–	–	–
T14	–	–	–	–
T15	2	UML model	C++	✗
T16	2	Other	C++	✓
T17	1	Other	C, C++	✗
T18	Other	–	C, C++, C##, ActionScript, Delphi, J, PHP, Python, VB, VHDL, SystemC, Other	✓
T19	1	Other	C++	✓

Table 13 Provided evidence and quality assessment scores (RQ3), identified limitations (RQ4)

Study	Research method	Type of evidence	Type of systems for evidence	Identified limitations
P01	V	EL	S, C, MR, MAN, COM	TO, E
P02	V	E	S	EP, TO, EV
P03	E	EXSI	MR	O, ECO, O, Other
P04	V	E	COM	E, EP, TO
P05	V	EL	MR, CE, C	E, O, EP, EV
P06	E	EVI	COM	E, O, TO
P07	V	E	B	E, TO
P08	V	–	–	O, EP, TO
P09	V	E	C	TO, AN
P10	V	EL	C	TO, E, AN, RE
P11	E	EXS	SE	–
P12	V	EXS	SE	E
P13	E	EXI	SE	TO, E
P14	V	E	MC	EV, E
P15	E	EXI	A	–
P16	E	EXI	MC, COM	AN, EV
P17	V	EXI	C	TO
P18	V	E	L	ECO
P19	V	E	S	PS
P20	V	E	MC	E
P21	E	EI	MAN	AN, RE
P22	E	EXS	COM	TO
P23	E	EI	MAN, SE, S	PS, ECO
P24	V	E	IS	AN, E
P25	V	E	MR	MC
P26	V	E	C	E, ECO, PI
P27	E	EXSI	C	E, PI, TO
P28	V	E	A	O
P29	V	E	COM	E
P30	V	E	C	E
P31	V	EXS	A	S, EV, E
P32	V	EXI	B	EV, R
P33	V	E	CE	R, MC
P34	V	EXS	S	EC
P35	V	EXS	MR	AN
P36	V	E	C	E, TO, T
P37	V	E	IS	E, TO
P38	V	EL	W	AN, E
P39	V	EXSI	MC	T
P40	E	EXS	S	TO, T, EV
P41	V	EXSI	COM	AN, RE
P42	V	E	MC	AN
P43	V	E	MC	–
P44	V	E	CE	–
P45	V	EXS	IS, L	EV, TO
P46	V	E	S	TO, E
P47	V	EXS	W	EV, EP
P48	V	E	MR	–
P49	V	E	S	MC, E
P50	V	E	S	–
P51	V	E	C	E, EP
P52	V	E	CE	–
P53	V	E	B	E, EV
P54	V	E	B	TO, Other
P55	V	E	MC	TO, P
P56	E	EI	IS	E
P57	V	E	MR	E, T
P58	V	EXS	MA	EV
P59	V	E	MAN	E, O
P60	V	EXI	COM	TO
P61	V	E	B	E
P62	V	EXS	S	E
P63	V	E	MAN	EP, E
T01	–	–	–	AN
T02	–	–	–	–
T03	–	–	–	–
T04	–	–	–	–
T05	–	–	–	AN, E
T06	–	–	–	–
T07	–	–	–	–
T08	–	–	–	–
T09	–	–	–	–
T10	–	–	–	–
T11	–	–	–	–
T12	–	–	–	–
T13	–	–	–	–
T14	–	–	–	AN
T15	–	–	–	MC
T16	–	–	–	E
T17	–	–	–	–
T18	–	–	–	–
T19	–	–	–	–

Table 14 Quality assessment scores (RQ3)

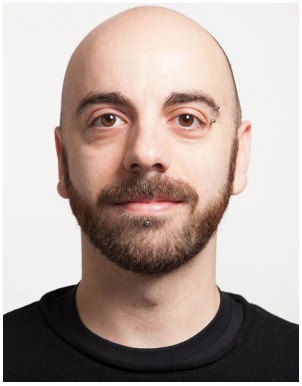
Study	Q1	Q2	Q3	Q4	Q5	Q6	Total
P01	1	1	1	1	0	0.5	4.5
P02	1	0.5	0.5	0.5	0	0	2.5
P03	1	1	1	1	0.5	0.5	5
P04	1	1	0.5	1	0	0	3.5
P05	1	0.5	1	1	0	1	4.5
P06	1	1	1	1	1	0.5	5.5
P07	0.5	0	0.5	0.5	0	0	1.5
P08	1	1	0.5	1	0.5	0.5	4.5
P09	1	0.5	1	1	0	0	3.5
P10	1	0	1	1	0	0	3
P11	1	0	0.5	1	0	0	2.5
P12	1	0	1	1	0	0	3
P13	1	1	1	1	0	0	4
P14	1	0	0.5	1	0	0	2.5
P15	1	0.5	0.5	1	0	0	3
P16	1	1	1	1	0.5	0	4.5
P17	0.5	0	0.5	1	0	0	2
P18	1	0	0.5	1	0	1	3.5
P19	1	0	0	1	0	0.5	2.5
P20	1	0	0.5	1	0	0.5	3
P21	1	1	1	1	0	0.5	4.5
P22	0.5	0.5	0	1	0	0	2
P23	1	1	1	1	1	1	6
P24	1	0.5	1	1	0	0.5	4
P25	1	0	1	1	0	0	3
P26	1	0	0	0.5	0	1	2.5
P27	1	1	1	1	1	0	5
P28	0.5	0	0.5	0.5	0	0	1.5
P29	1	0	0.5	0.5	0	0	2
P30	1	0.5	1	1	0.5	0	4
P31	1	0.5	0.5	1	0	1	4
P32	1	0.5	0	1	0	1	3.5
P33	0.5	0	0	0.5	0	0	1
P34	1	1	1	1	0	0	4
P35	1	1	1	1	0	0	4
P36	1	0	0.5	1	0	0.5	3
P37	0.5	0	0.5	1	0	1	3
P38	1	0.5	0.5	1	0	0	3
P39	1	0.5	1	1	0	0.5	4
P40	1	0.5	1	1	0	0	3.5
P41	1	1	1	1	0	0.5	4.5
P42	1	0	0.5	1	0	0	2.5
P43	0.5	0	0.5	1	0	0	2
P44	1	0	0	1	0	0	2
P45	1	0	0.5	1	0	0.5	3
P46	0.5	0	1	1	0.5	1	4
P47	1	0	1	1	0	0	3
P48	1	0	0.5	0.5	0	0	2
P49	1	0	0	1	0	0.5	2.5
P50	1	0	0.5	1	0	0	2.5
P51	1	0	0	1	0	0.5	2.5
P52	1	0.5	0.5	0.5	0	0	2.5
P53	1	0	0.5	1	0	0	2.5
P54	1	0.5	0.5	0.5	0	0	2.5
P55	0.5	0	0.5	1	0	0	2
P56	1	1	1	1	1	0	5
P57	0.5	0	0.5	1	0	0	2
P58	0.5	0	0.5	1	0	0	2
P59	1	0.5	0.5	0.5	0	0	2.5
P60	1	0.5	1	1	0	0.5	4
P61	1	0.5	1	1	0.5	1	5
P62	0.5	0	0	1	0	1	2.5
P63	0	0	0.5	1	0	0.5	2
T01	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T02	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T03	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T04	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T05	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T06	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T07	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T08	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T09	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T10	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T11	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T12	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T13	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T14	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T15	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T16	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T17	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T18	N/A	N/A	N/A	N/A	N/A	N/A	N/A
T19	N/A	N/A	N/A	N/A	N/A	N/A	N/A

References

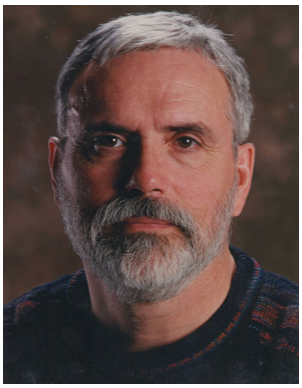
1. Abouzahra, A., Bézivin, J., Del Fabro, M.D., Jouault, F.: A practical approach to bridging domain specific languages with UML profiles. In: Proceedings of the Best Practices for Model Driven Software Development at OOPSLA, vol. 5. Citeseer (2005)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Agresti, A., Kateri, M.: Categorical Data Analysis. Springer, Berlin (2011)
4. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3), 515–526 (2006)
5. Ali, M.S., Babar, M.A., Chen, L., Stol, K.-J.: A systematic review of comparative evidence of aspect-oriented programming. Inf. Softw. Technol. **52**(9), 871–887 (2010)
6. Ali, N.B., Petersen, K.: Evaluating strategies for study selection in systematic literature studies. In: International Symposium on Empirical Software Engineering and Measurement. ACM (2014)
7. Aljer, A., Devienne, P., Tison, S., Boulanger, J.-L., Mariano, G.: BHDL: Circuit design in B. In: Proceedings. Third International Conference on Application of Concurrency to System Design, 2003, pp. 241–242. IEEE (2003)
8. Charfi, A., Mraidha, C., Gérard, S., Terrier, F., Boulet, P.: Toward optimized code generation through model-based optimization. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1313–1316 (2010)
9. Ciccozzi, F.: Unicomp: a semantics-aware model compiler for optimised predictable software. In: International Conference on Software Engineering (ICSE) 2018—New Ideas and Emerging Results (NIER), May 2018. UML, Alf, fUML, compilation, model-driven engineering, predictability, semantics
10. Ciccozzi, F., Cicchetti, A., Sjödin, M.: Round-trip support for extra-functional property management in model-driven engineering of embedded systems. Inf. Softw. Technol. **55**(6), 1085–1100 (2013)
11. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: an open-source tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, pp. 359–364. Springer, Heidelberg (2002)
12. Cohen, J.: Weighted kappa: nominal scale agreement provision for scaled disagreement or partial credit. Psychol. Bull. **70**(4), 213 (1968)
13. Corbin, J.M., Strauss, A.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Thousand Oaks (2014)
14. Cruzes, D.S., Dybå, T.: Research synthesis in software engineering: a tertiary study. Inf. Softw. Technol. **53**(5), 440–455 (2011)
15. Di Francesco, P., Malavolta, I., Lago, P.: Research on architecting microservices: trends, focus, and potential for industrial adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA), pp. 21–30. IEEE (2017)
16. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. In: Mazzara, M., Meyer, M. (eds.) Present and Ulterior Software Engineering, pp. 195–216. Springer, Cham (2017)
17. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: a systematic review. Inf. Softw. Technol. **50**(9), 833–859 (2008)
18. Franzosi, R.: Quantitative Narrative Analysis, vol. 162. Sage, Thousand Oaks (2010)
19. Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P.: Variability in software systems: systematic literature review. IEEE Trans. Softw. Eng. **40**(3), 282–306 (2014)
20. Gotti, S., Mbarki, S.: UML executable: a comparative study of UML compilers and interpreters. In: 2016 International Conference on Information Technology for Organizations Development (IT4OD), pp. 1–5 (March 2016)
21. Grandy, H., Bischof, M., Stenzel, K., Schellhorn, G., Reif, W.: Verification of Mondex electronic purses with KIV: from a security protocol to verified code. In: Woodcock, J. (ed.) FM 2008: Formal Methods, pp. 165–180. Springer (2008)
22. Greenhalgh, T., Peacock, R.: Effectiveness and efficiency of search methods in systematic reviews of complex evidence: audit of primary sources. BMJ **331**(7524), 1064–1065 (2005)
23. Hrischuk, C., Rolia, J., Woodside, C.M.: Automatic generation of a software performance model using an object-oriented prototype. In: Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1995. MASCOTS'95, pp. 399–409. IEEE (1995)
24. Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. **89**, 144–161 (2014)
25. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: Proceedings of ICSE. ACM (2011)
26. Iqbal, M.Z., Arcuri, A., Briand, L.: Environment modeling and simulation for automated testing of soft real-time embedded software. Softw. Syst. Model. **14**(1), 483–524 (2015)
27. Kitchenham, B., Brereton, P.: A systematic review of systematic review process research in software engineering. Inf. Softw. Technol. **55**(12), 2049–2075 (2013)
28. Kitchenham, B.A., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, Keele University and University of Durham (2007)
29. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained—The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, Reading (2003)
30. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics **33**, 159–174 (1977)
31. Laurent, Y., Bendraou, R., Gervais, M.-P.: Executing and debugging UML models: an fUML extension. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1095–1102. ACM (2013)
32. Lee, E.A., Seshia, S.A.: Introduction to Embedded Systems: A Cyber-Physical Systems Approach. MIT Press, Cambridge (2011)
33. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2013)
34. MARTE profile. http://www.omg.org/spec/MARTE/1.1/. Latest access: 20 Nov 2017
35. Mayerhofer, T., Langer, P., Wimmer, M., Kappel, G.: xMOF: Executable DSMLs based on fUML. In: Proceedings of SLE (2013)
36. Mellor, S.J., Tockey, S., Arthaud, R., Leblanc, P.: An action language for UML: proposal for a precise execution semantics. In: Bezivin, J. (ed.) The Unified Modeling Language.UML98: Beyond the Notation, pp. 307–318. Springer (1998)
37. Meyes, S.: The Most Important C++ Software...*Ever* (2006)
38. Montesi, F., Guidi, C., Zavattaro, G.: Composing services with JOLIE. In: Fifth European Conference on Web Services, 2007. ECOWS'07, pp. 13–22. IEEE (2007)
39. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, pp. 68–77, Swinton, UK (2008) British Computer Society
40. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. **64**, 1–18 (2015)
41. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)

42. Potter, B., Till, D., Sinclair, J.: An Introduction to Formal Specification and Z. Prentice Hall PTR, Englewood Cliffs (1996)
43. Rodgers, M., Sowden, A., Petticrew, M., Arai, L., Roberts, H., Britten, N., Popay, J.: Testing methodological guidance on the conduct of narrative synthesis in systematic reviews effectiveness of interventions to promote smoke alarm ownership and function. Evaluation **15**(1), 49–73 (2009)
44. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006)
45. Selic, B.: The less well known UML. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) Formal Methods for Model-Driven Engineering. Volume 7320 of Lecture Notes in Computer Science, pp. 1–20. Springer, Berlin (2012)
46. Steele, G.: Common LISP: The Language. Elsevier, London (1990)
47. Tatibouët, J., Cuccuru, A., Gérard, S., Terrier, F.: Formalizing execution semantics of UML profiles with fUML models. In: Model-Driven Engineering Languages and Systems, pp. 133–148. Springer (2014)
48. Tatibouët, J., Cuccuru, A., Gérard, S., Terrier, F.: Formalizing execution semantics of UML profiles with fUML models. In: Proceedings of MODELS, pp. 133–148 (2014)
49. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, p. 38. ACM (2014)
50. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, pp. 38:1–38:10. ACM, New York (2014)
51. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Computer Science. Springer, Berlin (2012)
52. Zhang, H., Babar, M.A.: Systematic reviews in software engineering: an empirical investigation. Inf. Softw. Technol. **55**(7), 1341–1354 (2013)
53. Zurowska, K., Dingel, J.: A customizable execution engine for models of embedded systems. In: Roubtsova, E., McNeile, A., Kindler, E., Gerth, C. (eds.) Behavior Modeling—Foundations and Applications, pp. 82–110. Springer, Berlin (2015)

Federico Ciccozzi is an Associate Professor (Docent in computer science) at Mälardalen University, Västerås, Sweden, where he is involved in model-based engineering for embedded systems and industrial software engineering groups. His research focuses on several aspects of model-driven engineering (MDE) and component-based software engineering (CBSE) for the development of complex (often embedded) systems based on domain-specific modeling languages (DSMLs), both UML- and EMF-based. Among them, he specializes in: definition of DSMLs, automatic model manipulations through transformations, and system properties preservation. Moreover, he conducts research in the area of multiparadigm modeling, model versioning, (co)evolution and synchronization, and the application of MDE and CBSE techniques to mobile multi-robot systems. He has (co-)authored over 60 publications in international journals, international conference, and workshop proceedings.



Ivano Malavolta is Assistant Professor at the Vrije Universiteit Amsterdam, The Netherlands. His research focuses on software architecture, model-driven engineering (MDE), and mobile-enabled systems, especially how MDE techniques can be exploited for architecting complex and mobile-enabled software systems at the right level of abstraction. He is program committee member and reviewer of international conferences and journals in his fields of interest. He is applying empirical methods to assess practices and trends in the field of software engineering. He authored more than 70 papers in international journals and peer-reviewed international conferences proceedings. He received a Ph.D. in computer science from the University of L'Aquila in 2012. He is a member of ACM and IEEE. More information is available at http://www.ivanomalavolta.com.



Bran Selic Mag.Ing., is President and Founder of Malina Software Corp., a Canadian company providing IT consulting and training services. He is also Director of Advanced Technology at Zeligsoft (2009) Limited in Canada and a Visiting Scientist at Simula Research Laboratories in Norway. On the academic side, Bran is currently an adjunct professor of software engineering at Monash University (Australia), a visiting researcher at the University of Sydney (Australia), and an invited industry lecturer at INSA in Lyon (France). In the course of his 45-year career in industry, he has pioneered the development of model-based methods, technologies, and standards for the real-time and embedded systems domain.