UML diagram synthesis techniques: a systematic mapping study

Damiano Torre[§,¶], *Yvan Labiche*[§], *Marcela Genero*[¶], *Maria Teresa Baldassarre*[β], *Maged Elaasar*[§]

[§]Carleton University, SQUALL Research Group, Canada
[¶]University of Castilla-La Mancha, ALARCOS Research Group, Spain
[β]University of Bari Aldo Moro, SERLAB Research Group, Italy

dctorre@sce.carleton.ca, yvan.labiche@carleton.ca, marcela.genero@uclm.es, mariateresa.baldassarre@uniba.it, melaasar@gmail.com

ABSTRACT

Context: UML software development relies on different types of UML diagrams, which must be consistent with one another. UML Synthesis techniques suggest to generate diagram(s) from other diagram(s), thereby implicitly suggesting that input and output diagrams of the synthesis process be consistent with one another.

Objective: Our aim is to provide a comprehensive summary of UML synthesis techniques as they have been described in the literature to date to then collect UML consistency rules, which can then be used to verify UML models. Method: We performed a Systematic Mapping Study by following well-known guidelines. We selected 14 studies by means of a search with seven search engines executed until January, 2018. Results: Researchers have not frequently published papers concerning UML synthesis techniques since 2004. We present a set of 47 UML consistency rules collected from the different synthesis techniques analyzed. Conclusion: Although UML diagrams synthesis doesn't seem to be an active line of research, it is relevant since synthesis techniques rely on enforcing diagram consistency, which is an active line of research. We collected consistency rules which can be used to check UML models, specifically to verify if the diagrams of a model are consistent with one another.

KEYWORDS

UML, UML synthesis techniques, Systematic mapping study, UML consistency rules, Model consistency checking.

1. INTRODUCTION

Model Driven Architecture (MDA) [1] promotes a set of transformations between successive models starting from requirements to analysis, to design, to implementation, and finally to deployment [2]. Much attention has been paid to MDA in academia and industry in recent years [3], which has resulted in models gaining even more importance in software development. The Unified Modeling Language (UML) [4] is the de facto standard formalism for object-oriented modeling and documentation [5]. It is the privileged modeling language when implementing MDA. The architecture of UML is based on a four-layer meta-model structure, and it provides 14 diagram types [4] that can be used to describe a system from different perspectives (e.g., structure, behavior) and/or abstraction levels (e.g., analysis, design). This helps deal with the complexity of system specification and distribute its responsibilities among different stakeholders, among other benefits. Since the various UML diagrams describe different aspects of one, and only one, software system under development, they are not independent but strongly depend on each other in many ways. In other words, the diagrams must be consistent [6]. Dependencies between diagrams can become so intricate that it is sometimes even possible to synthesize one diagram from other ones [7, 8]. The term "synthesis" has traditionally been used to describe the automatic construction of a program or a behavioral model from formal requirements [9]. In this paper, the accepted basic definition of synthesis is that of generating a UML diagram of one type from one or more of the remaining 13 types [7, 8]. Support for synthesizing one UML diagram from other diagrams can provide the designer with significant help, thus speeding up the design process, decreasing the risk of errors, and guaranteeing consistency among the various diagrams in the model [10]. Although many researchers have proposed, explicitly or implicitly, techniques to synthesize different types of UML diagrams, no well-accepted and as complete as possible set of UML synthesis techniques has been described to date [11, 12]. Our main research question is therefore: What is the current state-of-the-art in UML diagram synthesis? This endeavor additionally fits into another research activity we are conducting, which focuses on the general notion of consistency between UML diagrams that describe one software system [6, 13-16]. Indeed, while synthesizing one diagram from others, a synthesis technique enforces some consistency between the source diagram(s) and the generated diagram. However, synthesis techniques may or may not explicitly specify a set of consistency rules. A further objective of this paper is therefore to discover the consistency rules that result from UML diagram synthesis techniques. To achieve these goals, we have performed a Systematic Mapping Study (SMS) [17], following the guidelines of Kitchenham and Charters [18] and Petersen et al. [19]. This is a research method that provides an objective procedure with which to identify the amount of existing research related to a research question. Performing an SMS has several benefits [20]: it provides a starting point for a research topic, and in the longer term it provides a body of knowledge for the next generation of researchers.

The SMS protocol adopted in this work [18] is illustrated in the next section (section 2). We then discuss the results (section 3). The paper ends with a discussion of our conclusions (section 4).

MiSE'18, May 27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5735-7/18/05...$15.00
https://doi.org/10.1145/3193954.3193957

2. SMS– PLANNING AND EXECUTION

In this section, we present the main components of the planning of our SMS [18], which are: the specification of the research questions that the study aims to answer (see below); the strategy followed to find existing research papers on the topic (section 2.1); the procedures followed to select (or reject) papers for further investigation (section 2.2); and the procedures that are used to extract data from the selected papers in order to answer the research questions (section 2.3). We then discuss the execution of the SMS (section 2.4). The underlying motivation for the research questions was to determine the current state-of-the-art as regards to UML synthesis techniques and collect the corresponding set of UML diagram consistency rules. We therefore considered posing the following seven research questions (RQ#) along with their main motivations (MM#): **RQ1)** Which versions of UML are used by researchers in their synthesis techniques? **MM1**: To understand which versions of UML are used in UML synthesis techniques and whether there are any differences between UML versions (possibly owing to differences between metamodel versions). **RQ2**) Which types of UML diagrams are used as the source and target of synthesis techniques? **MM2**: To discover the UML diagrams that research has focused on in order to reveal the UML diagrams that are most frequently considered in this topic, and perhaps those that lack attention. **RQ3)** Which technologies are used to implement synthesis techniques? **MM3**: To find the technologies that are used to implement UML diagram synthesis. From the outset, it is possible to think of, for instance, model transformation technologies and ad hoc transformation algorithms. **RQ4)** Are the UML synthesis techniques automatic, semi-automatic, or manual? **MM4**: To discover whether the UML diagram synthesis techniques are automated or require human input. **RQ5)** What types of research papers report on UML synthesis techniques? **MM5**: To determine whether the field is generally more applied or consists of more basic research, as evidenced when classifying papers according to a well-known taxonomy [21]. **RQ6)** What are the UML consistency rules involved in the UML synthesis techniques? **MM6**: To find the UML consistency rules involved in the context of UML synthesis and to assess the state of the field. **RQ7**) Which types of UML consistency problems are tackled in the existing UML synthesis techniques? **MM7**: To find the types of consistency problems [22] tackled in the consistency rules involved in UML synthesis techniques: 1) horizontal, vertical and evolution consistency; 2) syntactic and semantic consistency; 3) observation and invocation consistency; and assess the state of the field as regards that dimension of UML diagram consistency.

2.1. Search strategy

Conducting a search for research papers using search engines requires the identification of a search string, and the specification of the parts of research papers in which the search string will be sought (the search fields). We identified our search string by following the procedure of Brereton et al. [23], which is composed of five steps: 1) Define the major terms; 2) Identify alternative spellings, synonyms or related terms for major terms; 3) Check the keywords in any relevant paper that may already be available to consolidate the list of terms; 4) Use the Boolean OR to incorporate alternative spellings, synonyms or related terms; 5) Use the Boolean AND to link the major terms. In our case, the major search terms were "UML" and "Synthesis" and the alternative spellings, synonyms or terms related to the major terms were: 1) UML (uml OR unified modeling language OR unified modelling language); 2) Synthesis (synthesis diagram). When selecting the search string, we considered various alternatives with the following objective in mind: we were interested in collecting synthesis techniques, or in other words, in identifying synthesis algorithms in order to precisely understand them and identify the consistency rules they (implicitly) enforce. Other search strings were experimented with, but are not discussed in this paper due to space limitations. Of all the alternative search strings in the set, we selected the following one as it allowed us to retrieve, the largest number of papers focusing on UML diagram synthesis:

((uml OR unified modeling language OR unified modelling language) AND (Synthesis Diagram)).

The search was limited to electronic papers and in particular those that were published in peer-reviewed journals, international conferences and workshops only in English language. We did not establish any restriction with regard to the publication year, except that we stopped with papers published in January 2018 at the latest. We used the above mentioned search string with the following seven search engines: Scopus, Google Scholar, CiteSeer, IEEE Digital Library, Science Direct, ACM Digital Library, and Inspec database. The searches were limited to title, keywords and abstract.

2.2. Selection procedure

Since a systematic, automated search in databases based on a search string may return papers that cannot be used to answer a set of research questions of interest in an SMS (e.g., in our case, a paper that mentions the synthesis of test cases from a UML sequence diagram), the set of papers collected from the search needs to be trimmed in a systematic manner. This is typically done by relying on so-called inclusion and exclusion criteria. Inclusion and exclusion criteria are based on the research questions and are piloted to ensure that they can be reliably interpreted and that they classify studies correctly [18]. In this section we discuss the inclusion and exclusion criteria we used. According to standard terminology in empirical software engineering, the set of research papers finally retained is referred to as primary studies [18]. The inclusion criteria were: 1) Electronic papers focusing on UML diagram synthesis; 2) Electronic papers written in English; 3) Electronic papers published in peer-reviewed journals, international conferences and workshops; 4) Electronic papers published until January, 2018; 5) Electronic papers which proposed UML synthesis techniques. The exclusion criteria were: 1) Electronic papers not focusing on UML diagram synthesis; 2) Electronic papers which were extended abstracts; 3) Duplicated electronic papers (e.g., returned by different search engines); 4) Electronic papers which discussed synthesis techniques between UML diagrams and other non-UML sources of data, such as requirements or source code.

2.3. Data extraction strategy

We extracted the data from the primary studies according to a number of criteria, which were directly derived from the seven research questions detailed in Section 2. Using each data extraction criterion required we read the full-text of each primary study. Once extracted, we recorded data on a spreadsheet that represented our data form. The following information was collected from each primary study:

1) Search engines: where the paper was found (see section 2.1);

2) Inclusion and exclusion criteria used (see section 2.2);

3) Data related to research questions (see section 2):

a) Which UML version was used;

b) What the UML synthesis techniques are;

c) The UML diagrams that were involved as input and output in synthesis techniques: Class (CD), Collaboration (COD), Use Case (UCD), Communication (COMD), State Machine (SMD), Sequence (SD), Protocol State Machine (PSMD), Object (OD), Interaction (ID), Activity (AD), Composite Structure (CSD), Timing (TD), Interaction Overview (IOD), and Deployment (DD) Diagram;

d) Tool support: 'Automatic' signifies that the UML synthesis techniques were fully-automated, i.e., supported by an implemented and working tool; 'Semi-automatic' means that the synthesis techniques were partially automated (for instance when the synthesis of a UML model needs a user's support for the process to be completed); 'Manual' means that the UML synthesis techniques were not supported by any implemented and automatic tool;

e) Type of research used in the primary study, for which we used the following classification [21]: *i)* An *Evaluation Research* (ER) paper investigates techniques that are implemented in practice, and reports on their evaluation. Such a paper shows how the technique is implemented in practice (solution implementation) and what the consequences of the implementation are in terms of benefits and drawbacks (implementation evaluation). *ii)* A *Proposal of Solution* (PS) paper proposes a solution to a problem and argues for its relevance, without a full-blown validation. *iii)* A *Validation Research* (VR) paper investigates the properties of a solution that has not yet been implemented in practice. *iv)* A *Philosophical Paper* (PP) sketches a new way of looking at things, a new conceptual framework, etc. *v)* An *Opinion Paper* (OP) contains the author's opinion about what is wrong or good about something, how something should be done, etc. *vi)* A *Personal Experience Paper* (PEP) places more emphasis on what than on why. The experience may concern one project or more, but it must be the author's personal experience;

f) The consistency rules implicitly enforced by the UML synthesis technique(s) (see Appendix A).

g) UML consistency dimensions, as discussed in the [22]: *i) Horizontal, Vertical and Evolution Consistency*: Horizontal consistency, also called intra-model consistency, refers to consistency between different diagrams at the same level of abstraction (e.g., class and sequence diagrams during analysis) in a given version of a model [24]; Vertical Inconsistency, also called inter-model consistency, refers to consistency between diagrams at different levels of abstraction (e.g., analysis vs. design) in a given version of a model [25]; Evolution consistency refers to consistency between diagrams of different versions of a model in the process of evolution [24]. *ii) Syntactic versus Semantic consistency*: Syntactic consistency ensures that the elements involved in the synthesis of two different diagrams conforms in the same way to the abstract syntax specified by the meta-model. Semantic consistency requires diagrams to be semantically compatible [25], and is not restricted to behavioral diagrams. Semantic consistency applies at one level of abstraction (with horizontal consistency), at different levels of abstraction (vertical consistency), and during model evolution (evolution consistency) [26]. *iii) Observation versus Invocation consistency*: Observation consistency requires an instance of a subclass to behave like an instance of its superclass, when viewed according to the superclass description [27]." Invocation consistency requires an instance of a subclass of a parent class to be used wherever an instance of the parent is required [28].

2.4. Execution

The execution for this SMS with the seven search engines began in September 2017 and was completed in January 2018. In this section we present the use of the search string with these engines and the selection of primary studies according to the inclusion/exclusion criteria previously described. In order to document the review process with sufficient details [18], we describe the multi-phase process of the four sub-phases employed: SP1) First sub-phase: the search string was used to search in the seven search engines, as mentioned earlier. SP2) Second sub-phase: we deleted duplicates. SP3) Third sub-phase: we obtained an initial set of studies by reading the title, abstract and keywords of all the papers obtained after SP2 while enforcing the inclusion and exclusion criteria. When this was not sufficient to decide whether or not to include or exclude a paper, we checked the full-text of the paper. SP4) Fourth sub-phase: the papers from SP3 were read in their entirety while reapplying the exclusion criteria. This resulted in the final set of 14 primary studies. Table 1 breaks down the number of papers we have found into sub-phases. We eventually collected 14 primary studies for further analysis [7, 8, 10, 29-39]. These were then analyzed according to the criteria detailed in section 2.2 and data was recorded in a spreadsheet.

Table 1: Summary of Primary Study Selection

Sub phase	IEEE	Inspec	ACM	Google Scholar	Cite Seer	Science Direct	Scopus	Tot.
SP1	33	98	18	14	25	6	81	275
SP2	20	20	7	4	13	3	41	108
SP3	8	7	6	0	4	0	3	28
SP4	4	3	3	0	3	0	1	14

3. SYSTEMATIC MAPPING STUDY – RESULTS

A quantitative summary of the results for research questions RQ1-RQ5 and RQ7 is presented in Table 2. The results of RQ6 are presented in Appendix A. More details are provided in sections 3.1 to 3.7. We then discuss some of the main findings in section 3.8. Percentages were rounded to the nearest integer value.

3.1. UML version (RQ1)

Although UML 2.x has an improved semantics in comparison to UML 1.x, which could help devise diagram synthesis techniques, we did not find more synthesis techniques for UML 2.x (five papers, i.e., 35%) than for UML 1.x (9 papers, i.e., 64%) during the period ranging between 1999—2016 (period of publication of the 14 primary studies): Table 2. It is interesting to note that we found more papers on the synthesis of UML 1.x diagrams than on UML 2.x in a shorter period of time (four years in the case of UML 1.x versus 13 years in the case of UML 2.x). No UML version was reported in a paper [34], but as it was published in 2003 which is the same year of publication as the UML version 2.0 [4], we classified it as a UML version 1.4 paper assuming the work was conducted prior to publication of the paper, i.e., prior to the publication of the UML 2.0.

Table 2: SMS results at a glance

Research question	Possible Answer		Result		Reference
			# Papers	Percentage	
RQ1: UML versions	UML 1.3		7	50%	[8, 10, 32, 33, 35, 37, 38]
	UML 1.4		2	14%	[7, 34]
	UML 2.0		2	14%	[31, 36]
	UML 2.1.		1	7%	[29]
	UML 2.1.2		1	7%	[30]
	UML 2.2		1	7%	[39]
RQ2: UML diagrams	Diagram to start the synthesis	IOD	1	6%	[29]
		SMD	2	13%	[7]
		AD	1	6%	[36]
		SD	11	69%	[7, 8, 10, 30-32, 34, 35, 37-39]
		COD	1	6%	[33]
	Diagram to synthesize	CD	3	19%	[7, 8, 35]
		SMD	11	69%	[7, 10, 29, 31-34, 36-39]
		AD	1	6%	[30]
		SD	1	6%	[7]
RQ3: UML Synthesis techniques	IOD → SMD		1	6%	[29]
	SD → SMD		8	50%	[34, 37, 38] [7, 10, 31, 32, 39]
	SD → CD		2	13%	[8, 35]
	AD → SMD		1	6%	[36]
	SD → AD		1	6%	[30]
	COD → SMD		1	6%	[33]
	SMD → SD		1	6%	[7]
	SMD → CD		1	6%	[7]
RQ4: Research methods	Evaluation Research		1	7%	[8]
	Validation Research		1	7%	[7]
	Proposal of Solution		12	86%	[10, 29-39]
RQ5: Support	Automatic		12	86%	[7, 8, 10, 29, 32-39]
	Manual		2	14%	[30, 31]
RQ7: Type of rules	Semantic Consistency		41	87%	See Table 4 in Appendix A for details
	Semantic Consistency		6	13%	
	Horizontal Consistency		47	100%	

3.2. UML diagrams (RQ2)

Figure 1 summarizes the number of times each diagram of some synthesis has been found to be the source or target.

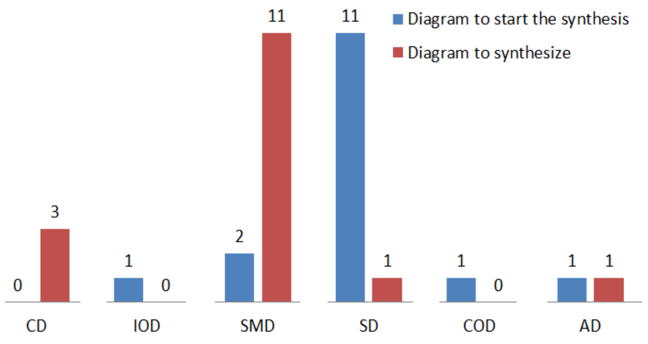

Figure 1: UML diagrams in Synthesis Techniques

The sequence diagram (SD) is by far the diagram type that is most frequently used for the synthesis of a diagram of another type (68%, 11 synthesis techniques). Conversely, the state machine diagram (SMD) is the most synthesized diagram type (68%, 11 techniques), followed by the class diagram (CD) (19%, three techniques). Overall, only half of the UML diagram types are involved in some sort of synthesis technique, and it is not entirely surprising that the diagram types involved in synthesis techniques are also identified as the most frequently used diagrams [40].

3.3. Technologies for synthesis techniques (RQ3)

In the 14 primary studies, we identified 16 different synthesis techniques, each of which involved one input diagram and one output diagram, which can be classified into eight different types of synthesis techniques: Figure 2 and Table 2.

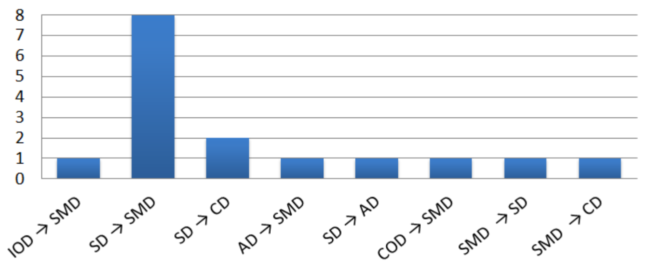

Figure 2: From → To Synthesis Techniques

The most frequently described synthesis technique type is sequence diagram to state machine diagram (SD→SMD) for a total of eight techniques (50%, out of 16 synthesis techniques). Each of the 16 synthesis techniques is summarized below, and we also discuss the technologies they rely on in order to perform the synthesis. Due to space limit, in this section we only describe 16 UML synthesis techniques considered in this paper. Whittle and Jayaraman [29] presented a technique to automatically synthesize hierarchical state machine for state dependent objects from Extended Interaction Overview Diagrams (EIODs), that is a three-layer structure, with a precise semantics, where the top layer defines use cases, the second layer defines scenarios, and the bottom layer defines sequence diagrams. We classify this synthesis technique as a mapping since, even though the authors precisely describe the mapping between elements of the EIOD to elements of hierarchical state machines, they did not provide a detailed algorithm or did not rely on some other technology (e.g., model transformation) to perform the synthesis. So far, research mostly focused on technical issues such as how to best derive SMD from SD and how to best ensure the correctness of this synthesis: half (five) of the UML synthesis techniques address this issue. The Minimally Adequate Synthesizer (MAS) [10] infers a SMD from a set of SDs, using an engineer-guided grammatical inference technique: the state machine is seen as describing an unknown grammar to be inferred, where the sequence diagrams are sentences. We classify this synthesis technique as grammatical inference. Ziadi and colleagues [31], very similarly to Whittle and Jayaraman [29], introduce an algebraic framework for such a synthesis: the authors define an algebraic representation of sequence diagrams, an algebraic representation of state machine diagrams, and then a precise mapping between the two. We classify this technique as algorithmic since the authors provide a detailed algorithm for the mapping. Schumann [32] synthesized state machines from sequence diagrams, where messages are annotated with OCL pre and post conditions, based on an earlier work of the author [37], which is itself based on an earlier work of Whittle [29] that does not account for interaction fragments. We classify this technique as algorithmic since the earlier paper provides a detailed algorithm for the synthesis. The Fujaba project [34] also includes this type of synthesis. It is based on grammatical inference as in MAS [10], and that we therefore also classify as grammatical inference. Selonen and colleagues also rely on a grammatical inference algorithm to synthesize a series of state machines from sequence diagrams [7].

Khriss and colleagues [33] proposed an incremental UML synthesis technique to generate SMDs of all the objects involved from a set of CODs. The technique employed to perform the synthesis is informally described and we therefore classify this paper as informal description. Whittle and Schumann in both [37],[38] present an algorithm which supports the design process by generating state machine diagrams (SMD) automatically from scenarios (SD). Krka and Medevidovic [39] propose component-aware triggered scenarios (caTS) operationalized through a synthesis algorithm which represents a step forward to triggered scenario languages. On the other hand, Eshuis and Gorp [36] present an automated synthesis approach for generating state machine diagrams from activity diagrams.

These different synthesis approaches [7, 10, 29, 31-34] are very similar in the sense that they recognize object interactions, provided in various, though similar forms (Extended Interaction Overview Diagrams, Sequence Diagrams, Collaboration Diagrams) as specifying legal sequences of messages objects can receive and from which the complete set of legal sequences of messages that objects can receive can be constructed (i.e., the state machine diagrams). Selonen and colleagues presented the synthesis of UML CDs from SDs in 2000 [35], which they extended in 2001 [8] and summarized, along with other synthesis techniques (see below) in 2003 [7]. The authors informally discussed how elements of sequence diagrams can be mapped to elements of a class diagram. We therefore classify this technique as informal description. Kang and colleagues [30] synthesized ADs from SDs thereby representing the flow of messages in sequence diagrams as activity diagrams. The mapping rules from SD to AD are precisely discussed and the overall synthesis is described with an algorithm. We therefore classify the technique as algorithmic. We already discussed two of the synthesis techniques that Selonon and colleagues summarized [7]. They also discuss the synthesis of CD from SMD, as well as the synthesis of SMD from SD following the work of Systä [41]. These are only summaries, which we classify as informal descriptions.

3.4. Research papers (RQ4)

As reported in Table 3, the primary studies can be classified as ER (Evaluation Research), VR (Validation Research), and PS (Proposal of Solution), contributing 7%, 7% and 86% of the total, respectively. The vast majority of the primary studies therefore propose a complete technical solution for the synthesis of UML diagrams. One paper was categorized as an evaluation research paper [8] and one other was categorized as a validation research paper [30].

3.5. Type of support (RQ5)

As described in Table 3, the synthesis techniques we collected are usually supported by a tool (86%, 12 papers). Only one of them [29] is integrated with its UCSIM UML CASE tool, a vendor-independent implementation that the authors intended to integrate with IBM Rational Software Modeler. The synthesis technique of Schumann [32] was implemented in Java and the author mentioned the intention to integrate the technique into the commercial UML tool Magic Draw. We were, however, unable to find more recent publications by this author since the original publication of the work in 2008 confirming that this actually happened. The synthesis technique of Selonen and colleagues [7, 8, 35] was implemented in an industrial software development environment by the same authors: the Nokia TED. Whittle and Schumann in both [37] and [38] present a prototype of their synthesis algorithm which has been implemented in Java. Maier and Zndorf [34] implemented their synthesis technique in the Fujaba environment, which is a free open source UML CASE tool environment, to provide roundtrip engineering support. Ziadi and colleagues [31] developed a prototype of their synthesis technique in Java. Mäkinen, and Systä [10] implemented their synthesis technique in their own prototype tool, whereas Krka and Medvidovic [39] illustrate a synthesis algorithm as a specification of their component-aware triggered scenario (caTS). Kang and colleagues [30] mentioned that they were planning to improve their synthesis technique by automating its manual steps. Khriss and colleagues similarly deferred complete automated support (in their case adding editors for the UML diagram types involved in their synthesis technique) to future work [33]. We were unable to find more recent publications by these authors discussing these aspects since the initial publications (2010 and 1998, respectively).

3.6. UML consistency rules (RQ6)

Since the different UML diagrams that are constructed during a specific software development specify different aspects of one, and only one system, these diagrams must be consistent with one another. This is true regardless of whether the diagrams are generated by hand (using a CASE tool for instance) or synthesized from other diagrams. The UML diagram synthesis techniques we have collected, therefore, enforce some kind of consistency between the input diagrams of the synthesis and the diagrams being synthesized. We finally collected a set of 47 UML consistency rules (from the different synthesis techniques–the primary studies) that are presented in Table 4 of Appendix A. No consistency rule was identified in some of the primary studies [34, 36-39]. Paper [35] presented the same nine rules as in paper [8] but with a new characteristic that can be used to synthesize class diagrams with operation description annotations.

3.7. Types of consistency rules (RQ7)

The results obtained for RQ7 show that all the 47 rules were Horizontal (100%) and the vast majority of them Syntactic (87%, 41 out of 47 rules). Moreover, we found 6 (13%) Semantic rules. We conjecture that the main reason for this is that syntactic rules are easier to specify than semantic rules and that the UML standard more formally specifies syntax than semantics, which hinders the specification of semantic rules. It may also be the case that semantic ones are specific to the way in which the UML notation is actually used, which is organization, project, or team specific, and are therefore seldom described in published manuscripts.

3.8. Summary

Table 3 shows some results regarding the seven research questions according to each of the 14 primary studies [7, 8, 10, 29-39].

It is interesting to note the absence of synthesis techniques published between 2004 and 2010. Having identified nine primary studies with an old version of UML (64%) [7, 8, 10, 32-35, 37, 38] emphasizes that the synthesis of UML diagrams has started to become relevant since the initial launch of the UML (it has been evolving since the second half of the 1990s [4]). Furthermore, the other five primary studies that rely on the more recent UML versions 2.x [29-31, 36, 39] prove that the new version of the metamodel did have an impact on research in this domain. Another interesting aspect is that 11 of the 16 techniques synthesize UML sequence diagrams (68%).

Observations about the UML diagrams involved in synthesis techniques show that the synthesis of state machine diagrams from a collection of scenarios (i.e. sequence diagrams) has received a lot of attention. Finally we have observed that the researchers who are most actively involved in UML synthesis techniques are Jon Whittle [29, 37, 38] and Petri Selonen [7, 8, 35], both with three publications.

Table 3: Summary of Synthesis and Support

Ref.	Research papers	UML version	Year	Diagram synthesized	Technique used	Automatic support
[29]	PS	2.1.1	2010	IOD to SMD	Synthesis Algorithm	YES
[8]	ER	1.3	2001	SD to CD	Two tr. approaches	YES
[35]	PS	1.3	2000	SD to CD	Synthesis Algorithm	YES
[10]	PS	1.3	2001	SD to SMD	Synthesis Algorithm	YES
[34]	PS	1.4	2003	SD to SMD	Synthesis Algorithm	YES
[31]	PS	2.0	2004	SD to SMD	Algebraic framework Synthesis Algorithm	YES
[32]	PS	1.3	2000	SD to SMD	Synthesis Algorithm	YES
[30]	PS	2.1.2	2010	SD to AD	Mapping Rules Synthesis Algorithm	NO
[37]	PS	1.3	2000	SD to SMD	Synthesis Algorithm	YES
[38]	PS	1.3	2000	SD to SMD	Synthesis Algorithm	YES
[39]	PS	2.2	2014	SD to SMD	Synthesis Algorithm	YES
[36]	PS	2.0	2016	AD to SMD	Automated Approach	YES
[7]	VR	1.4	2003	SD to SMD SMD to SD/CD	Synthesis Algorithm	YES
[33]	PS	1.3	1998	COD to SMD	Synthesis Algorithm	NO

4. CONCLUSIONS

This work presents the results obtained after applying the Systematic Mapping Study (SMS) protocol with the aim of identifying and evaluating the current approaches for synthesizing UML diagrams from other UML diagrams. To the best of our knowledge, no such mapping studies or reviews existed prior to this work. The SMS was carried out systematically following well-known guidelines [18]. A significant amount of space in this paper has been used to discuss how these guidelines were followed, since we felt that it was of the upmost importance to allow readers to precisely understand the results and conclusions, and to allow others to replicate our study or compare our results with others. From an initial set of 108 papers published in literature and extracted from seven scientific research databases, a total of 14 studies were eventually analyzed in depth, by following a precise selection protocol driven by seven research questions. We observe that (in no particular order of importance):

1) There is no commercially available UML CASE tool standard with which to run UML synthesis techniques between UML diagrams.

2) The most frequently used UML diagram type as a source for synthesis is the sequence diagram (68%), and the most synthesized diagram types are the state machine diagram (68%) followed by class diagram (19%). This is not entirely surprising since these are the most frequently used UML diagrams [40].

3) We did not find any synthesis techniques for Communication, Protocol State Machine, Object, Interaction, Composite Structure, Timing and Deployment Diagrams. This may mean: a) that no such synthesis really makes sense; b) that additional research on synthesis techniques involving all 14 UML diagrams is warranted; c) or that those UML diagrams are least understood or least interesting.

4) We presented a set of 47 UML consistency rules systematically collected from the 14 primary studies. Having identified such a set of rules may provide several benefits for the MISE community: it provides a reference for practitioners, educators, and researchers alike; it provides guidance as to which rules to use in each context; and finally it highlights which areas (i.e., what software development phase) are more developed and which need further work. Moreover, the set of rules could be a good input to the UML revision task force for inclusion in a forthcoming standard.

5) All the 47 identified rules were horizontal and most of them syntactic (a few semantic). We did not find any vertical, invocation, observation and evolution consistency rules. One reason for this could be the fact that the UML syntax is more formally described (by the UML metamodel) in the specification than semantics, and creating syntactic rules may therefore prove more feasible. These results show that consistency rules specified in the context of UML synthesis techniques focus only on UML horizontal consistency problems.

6) Synthesizing UML diagrams from other UML diagrams is not a very active line of research since the most recent papers were published in 2010 [29, 30], in 2014 [39] and only one in 2016 [36]. In addition, we only collected 14 primary studies. We wonder whether the fact that we did not find many primary studies is likely due to a change of terminology, i.e. the MDE community nowadays prefers the term “transformation” to the term “synthesis”. Nevertheless, considering that any UML synthesis technique needs to enforce consistency between the input UML diagram and the output synthesized UML diagram, this topic is a very mature and relevant topic [6].

Finally, thanks to the synthesis techniques presented in the papers collected, a UML model, and its UML diagrams attain a higher quality: increased consistency and correctness, because they are either produced or updated automatically, or are checked against each other using the synthesis techniques. We feel that the time is ripe for these UML synthesis techniques to be put through their paces and that in-depth studies as well as replications [42] on how they can most effectively support UML consistency rules should now be undertaken.

Appendix A

Table 4 lists the 47 horizontal UML consistency rules. The first column "S.T." (Synthesis Technique) presents the diagrams involved in the technique (for example SD to SMD, from a Sequence Diagram, a State Machine Diagram is synthesized). The third column indicates whether this is a Syntactic (Sy) or Semantic (Se) rule. The fourth column indicates the reference (Ref.) of the primary study paper from where a given rule was collected.

Table 4: Summary of UML Consistency Rules

S.T.	UML Consistency rules		Ref.
SD to SMD	A reception in the SD becomes an event in the SMD and emissions become actions.	Sy	[31]
	For a transition associated with a reception, the action part will be empty, and for transitions associated with actions, the event part will be empty.	Sy	
	SMDs generated from SDs will be sequences of states, and will contain a single junction state that corresponds to the state reached when all events situated on an object lifeline have been executed.	Sy	
	When an object does not participate in an interaction, the projection of an SD on this object's lifeline is the empty word	Sy	
	For a single life line within a SD, incoming messages are interpreted as events attached to certain transitions and outgoing messages as do-actions of certain states.	Se	[10, 38]
	Following the lifeline of a given object, reuse already existing transitions and states in the corresponding SMD for as long as possible. Otherwise new transitions and states are created.	Se	
	For each such possible start element, compute the number of subsequent messages that match subsequent SMD elements without the need to create new elements.	Sy	
	Map items in the message trace in SD to transitions and states in an SMD.	Sy	[7]
	Messages sent in SD are regarded as primitive actions associated with states in SMD.	Sy	
	Each message received in SD is mapped in a SMD transition.	Sy	
	A synthesized SMD is deterministic, i.e., there cannot be two similarly labeled leaving transitions in any particular state, unless their guards are mutually exclusive. Two applicable transitions cannot be satisfied simultaneously.	Sy	
	A completion transition and a labeled transition cannot leave the same state unless their guards differ.	Sy	
SD to CD	Generate a class for each classifier role with a distinctive class name. Transfer stereotypes, such as «actor» and «active» into the respective class.	Sy	[8]
	For each message, if an association between the base classes of the sending and receiving classifier roles does not exist, generate the corresponding association ends and an association representing the communication connection for the message in question, and link them together.	Se	
	For each class receiving a message, if an operation with the same name does not already exist, generate a new operation for the class. Mark the navigability of the association.	Sy	
	If there are arguments attached to the message, generate new parameters for the operation with corresponding basic types and type expressions. Add this operation to the corresponding class if it does not already have an operation with the same name and signature.	Sy	
	If the message is marked with a stereotype «signal» and a signal with the same name does not exist, generate a new signal. Associate the corresponding class with this signal.	Sy	
	If the message is marked with a stereotype «become», the sending and receiving classifier roles are equated (this rule assumes that an object cannot dynamically change its type).	Sy	
	If the message has an object that appears in the SD as an argument, we add an association between the corresponding classes of the sending object and argument object, if one does not already exist and the classes are not the same.	Sy	
	If the message is a return message, containing only a return value of some type, we conclude the return type of the operation of the preceding message.	Sy	
	When SDs contain contradictory information, for example an active and a passive object of the same class. In this case, the result of the transformation operation should be undefined.	Se	
IOD to SMD	Normal edges in an EIOD are interpreted as a weak sequential composition but taking into account flow final and final nodes.	Sy	[29]
	Parallel fork/join edges lead to orthogonal regions in the SMDs generated. Since fork/join edges are well nested, there is no ambiguity as regards deciding where the orthogonal regions should be placed.	Sy	
	For preempting and suspending edges, composite states are introduced into the SMDs generated.	Sy	
	Negative edges result in orthogonal regions in the SMDs generated. Negation is handled by monitoring for the negative events and transitioning to a special error state if they occur. The negative events being monitored are placed in an orthogonal region so that, if the sequence of negative events ever occurs (even with other events interleaved), then the error state will be entered.	Sy	
	If a node group is marked as having multiple concurrent nodes (i.e., it has an asterisk attached to it), then the EIODs semantics state that the node group can be replaced with a parallel fork/join edge in which the node group is repeated an undetermined number of times.	Sy	
SD to AD	A self call action of an object in a SD translates to an action of the corresponding participant of an AD.	Sy	[30]
	A message passing description in a SD is translated using the notations such as send signal, receive signal, object nodes, and transition between the signal node and the object node.	Sy	
	When a reference is used in a SD, we omit explicit reference notation and simply use a reference name. This should then be later expanded after the expansion of the reference of a SD is recursively translated into the AD and connected to the rest of the AD	Sy	
	When references are used, the reference type of a SD specification is translated into the interaction occurrence type of an AD.	Sy	
	The application of the alt operator of a SD can be transformed into an AD by using the synchronization bars, decision nodes and the merge nodes.	Sy	
	In order to transform a loop operator of a SD into an AD, setup, test, and body sections should be identified first. After finishing the body section, each process that participates in the body section goes back to the join node to check the condition again.	Sy	
	The par operator of a SD represents the parallel composition of multiple scenarios. By using the fork and join nodes of an AD, a participant can be split into multiple threads and joined together.	Sy	
COD to SMD	Create an SMD for every distinct class implied by the objects in the COD.	Se	[33]
	Introduce as state variables all variables which are not attributes of the objects of COD.	Sy	
	Create transitions for the objects from which messages are sent.	Sy	
	Create transitions for the objects to which messages are sent.	Sy	
	For all SMDs, put the set of transitions generated into correct	Sy	

	sequences, connecting them by states, split bars and merge bars.		
SMD to SD	Actions are messages sent or actions performed by the object whose behavior is described by the SMD. Event triggers of transitions are messages received by the object.	Sy	[7]
	Actions that are attached to transitions or states are transformed into corresponding actions of an interaction. These actions imply messages and classifier roles.	Sy	
	Signal events and call events associated with transitions or states are also mapped onto send actions and call actions, respectively.	Sy	
	The external stimuli can also be given as an SMD, thus defining the response of an external actor to the output of the system.	Sy	
SMD to CD	Signal events of transitions and states are mapped onto signals and ultimately onto behavioral features that define their context.	Sy	[7]
	Call events are mapped onto operations of classifiers.	Sy	
	Send actions and call actions are mapped onto signals and behavioral features, and operations, respectively.	Sy	
	Actions are also used to infer stimuli and corresponding links which are in turn mapped onto associations and association ends of classifiers.	Sy	
	The context of the state machine is mapped onto the classifier for which the behavior is defined.	Se	

ACKNOWLEDGEMENTS
This research has been partly funded by a Discovery grant of the Natural Sciences and Engineering Research Council of Canada and the SEQUOIA project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2015-63502-C3-1-R).

REFERENCES
[1] J. Mukerji and J. Miller. 2003. Overview and guide to OMG's architecture, Object Management Group http://www.omg.org/mda/.

[2] D. Thomas. 2004. MDA: Revenge of the modelers or UML utopia?, *IEEE Software,* vol. 21, pp. 15–17.

[3] M. Genero, A. M. Fernández-Saez, H. J. Nelson, G. Poels, and M. Piattini. 2011. A Systematic Literature Review on the Quality of UML Models, *Journal of Database Management,* vol. 22, pp. 46-70, July-September 2011.

[4] OMG. 2011. OMG Unified Modeling LanguageTM - Superstructure Version 2.4.1.

[5] T. Pender. 2003. *UML Bible (1st ed.)*. New York, NY, USA: John Wiley & Sons, Inc.

[6] D. Torre, Y. Labiche, and M. Genero. 2014. UML consistency rules: a systematic mapping study, In *Proceeding of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014)*, London, UK.

[7] P. Selonen, K. Koskimies, and M. Sakkinen. 2003. Transformations between UML diagrams, *Journal of Database Management,* vol. 14, pp. 37-55.

[8] P. Selonen, K. Koskimies, and M. Sakkinen. 2001. How to Make Apples from Oranges in UML, In *Proceeding of the 34th Annual Hawaii International Conference on System Sciences (HICSS '01)*.

[9] Z. Manna and R. J. Waldinger. 1971. Toward automatic program synthesis. Commun., *ACM* vol. 14, pp. 151-165.

[10] E. Mäkinen and T. Systä. 2001. MAS — an interactive synthesizer to support behavioral modelling in UML, In *Proceeding of the 23rd International Conference on Software Engineering (ICSE '01)*, Washington, DC, USA.

[11] A. M. Fernández-Sáez, M. Genero, D. Caivano, and M. R. V. Chaudron. 2016. Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments *Empirical Software Engineering,* vol. 21, pp. 212-259.

[12] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, D. Caivano, and I. Ramos. 2015. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: A family of experiments, *IST,* vol. 57, pp. 644-663.

[13] D. Torre, Y. Labiche, M. Genero, M. Elaasar, T. K. Das, B. Hoisl, *et al.* 2016. 1st International Workshop on UML Consistency Rules (WUCOR 2015): Post workshop report, *SIGSOFT Softw. Eng. Notes,* vol. 41, pp. 34-37.

[14] D. Torre. 2014. On collecting and validating UML consistency rules: a research proposal, In *Proceeding of the Doctoral Symposium at 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014)*, London, UK.

[15] D. Torre. 2015. On validating UML consistency rules, In *Proceeding of the 26th IEEE International Symposium on Software Reliability Engineering, Doctoral Symposium (ISSRE 2015)* Gaithersburg, MD, USA, pp. 50-60.

[16] D. Torre. 2016. Verifying the consistency of UML models, In *Proceeding of the 27th IEEE International Symposium on Software Reliability Engineering, Fast Abstract (ISSRE 2016)* Ottawa, Canada.

[17] H. Arksey and L. O'Malley. 2005. Scoping studies: towards a methodological framework, *International Journal of Social Research Methodology,* vol. 8.

[18] B. Kitchenham and S. Charters. 2007. Guidelines for performing systematic literature reviews in software engineering, Keele University.

[19] K. Petersen, S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update, *Information & Software Technology,* vol. 64, pp. 1-18.

[20] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham. 2008. Using mapping studies in software engineering, In *Proceeding of the Psychology of Programming Interest Group Workshop*, Lancaster University, pp. 195–204.

[21] R. Wieringa, N. Maiden, N. Mead, and C. Rolland. 2005. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion, *Requirements Eng.,* vol. 11, pp. 102-107.

[22] T. Mens, R. Van der Straeten, and J. Simmonds. 2005. A framework for managing consistency of evolving UML models, in *Software Evolution with UML and XML*, ed: IGI Publishing, 2005, pp. 1-30.

[23] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil. 2007. Lessons from applying the systematic literature review process within the software engineering domain, *Journal of Systems and Software,* vol. 80, pp. 571–583.

[24] Z. Huzar, L. Kuzniarz, G. Reggio, and J. L. Sourrouille. 2005. Consistency problems in UML-based software development, In *Proceeding of the International Conference on UML Modeling Languages and Applications*, Lisbon, Portugal, pp. 1-12.

[25] G. Engels, J. H. Hausmann, and R. Heckel. 2002. Testing the consistency of dynamic UML diagrams, In *Proceeding of the Integrated Design and Process Technology*, Pasadena, California.

[26] M. A. Ahmad and A. Nadeem. 2010. Consistency checking of UML models using Description Logics: A critical review, In *Proceeding of the 6th International Conference on Emerging Technologies* Islamabad, Pakistan, pp. 310-315.

[27] G. Engels, R. Heckel, and J. M. Küster. 2001. Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model, In *Proceeding of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Toronto, Ontario, Canada, pp. 272-286.

[28] G. Engels, J. M. Küster, R. Heckel, and L. Groenewegen. 2001. A methodology for specifying and analyzing consistency of object-oriented behavioral models, *Sigsoft Software Engineering Notes,* vol. 26, pp. 186-195, September 2001.

[29] J. Whittle and P. K. Jayaraman. 2010. Synthesizing hierarchical state machines from expressive scenario descriptions, *ACM Trans. Softw. Eng. Methodol.,* vol. 19.

[30] S. Kang, H. Kim, J. Baik, H. Choi, and C. Keum. 2010. Transformation Rules for Synthesis of UML Activity Diagram from Scenario-Based Specification In *Proceeding of the IEEE 34th Annual Computer Software and Applications Conference (COMPSAC '10)*, pp. 431-436.

[31] T. Ziadi, L. Helouet, and J.-M. Jezequel. 2004. Revisiting Statechart Synthesis with an Algebraic Approach In *Proceeding of the 26th International Conference on Software Engineering (ICSE '04)*.

[32] J. Schumann. 2000. Automatic debugging support for uml designs, In *Proceeding of the 4th International Workshop on Automated Debugging*.

[33] I. Khriss, M. Elkoutbi, and R. K. Keller. 1998. Automating the Synthesis of UML StateChart Diagrams from Multiple Collaboration Diagrams, In *Proceeding of the 1st International Workshop on The Unified Modeling Language: Beyond the Notation (UML '98)*, pp. 132-147.

[34] T. Maier and A. Zndorf. 2003. The Fujaba Statechart Synthesis Approach, In *Proceeding of the Workshop on Scenarios and State Machines (ICSE '03)*, Portland, Oregon, USA.

[35] P. Selonen and T. Systä. 2000. Scenario-based Synthesis of Annotated Class Diagrams in UML, In *Proceeding of the Workshop: Scenario-based round-trip engineering (OOPSLA '00)* Tampere University of Technology, Software Systems Laboratory, pp. 26-31.

[36] R. Eshuis and P. V. Gorp. 2106. Synthesizing object life cycles from business process models, *Software & Systems Modeling,* vol. 15, pp. 281–302.

[37] J. Whittle and J. Schumann. 2000 Generating statechart designs from scenarios, In *Proceeding of the 22nd international conference on Software engineering (ICSE '00)*.

[38] J. Schumann and J. Whittle. 2000. Automatic Synthesis of Agent Designs in UML, In *Proceeding of the 1st International Workshop on Formal Approaches to Agent-Based Systems (FAABS '00)*.

[39] I. Krka and N. Medvidovic. 2014. Component-Aware Triggered Scenarios, In *Proceeding of the 2014 IEEE/IFIP Conference on Software Architecture (WICSA '14)*.

[40] B. Dobing and J. Parsons. 2006. How UML is used, *ACM,* vol. 49, pp. 109-113.

[41] T. Systä. 2000. Incremental construction of dynamic models for object-oriented software systems, *Journal of Object-Oriented Programming,* vol. 13, pp. 18-27.

[42] M. T. Baldassarre, J. Carver, O. Dieste, and N. Juristo. 2014. Replication types: Towards a shared taxonomy, *ACM International Conference Proceeding Series*.