State Machine Flattening, a Mapping Study and Tools Assessment

Xavier Devroey, Maxime Cordy, Pierre-Yves Schobbens
PReCISE, University of Namur, Belgium
{firstname.name}@unamur.be

Axel Legay
INRIA Rennes Bretagne Atlantique, France
axel.legay@inria.fr

Patrick Heymans
PReCISE, University of Namur, Belgium
patrick.heymans@unamur.be

***Abstract*—State machine formalisms equipped with hierarchy and parallelism allow to compactly model complex system behaviours. Such models can then be transformed into executable code or inputs for model-based testing and verification techniques. Generated artifacts are mostly flat descriptions of system behaviour. *Flattening* is thus an essential step of these transformations. To assess the importance of flattening, we have defined and applied a systematic mapping process and 30 publications were finally selected. However, it appeared that flattening is rarely the sole focus of the publications and that care devoted to the description and validation of flattening techniques varies greatly. Preliminary assessment of associated tool support indicated limited tool availability and scalability on challenging models. We see this initial investigation as a first step towards generic flattening techniques and scalable tool support, cornerstones of reliable model-based behavioural development.**

***Keywords*—State machine; Flattening; Systematic Mapping Study**

I. Introduction

State machines are popular models of system behaviour. By providing them with a formal semantics, one can perform automated behavioural analysis (e.g. by model checkingor model-based testing) and code generation. In order to model complex systems in a concise and comprehensible manner, state machines have been equipped with various abstraction constructs such as hierarchy and parallelism [1]. Yet, abstraction comes with the cost of more elaborated semantics and potential ambiguities (e.g. in UML), thus preventing the direct use of automated analysis and generation tools.

Flattening [1] – a procedure that systematically transforms hierarchical state machines into state machines where all states are atomic – was proposed as an answer. It bridges succinct modelling with formal semantics and automated analysis, allowing to envision end-to-end model-driven validation chains for complex systems [2]. Flattening plays a pivotal role in behavioural analysis of software systems. Hence, its role in model-basel development and validation should be fully understood.

In spite of its importance and widespread use, there has been no systematic effort to categorize flattening approaches and their applicability. This paper is a first step in this direction. We examine almost 20 years of scientific literature and perform a systematic mapping study [3]. We follow the systematic approach used in the medical field [4], which is more appropriate for categorization purpose than systematic literature reviews [3], [4]. We nevertheless incorporated some relevant elements of systematic literature reviews as suggested by Petersen *et al.* [3]. After an initial search that returned 167 publications, 30 of them were finally considered as relevant for the mapping. Our mapping relies on 4 dimensions (also called facets) covering research purpose, input/output models or the type of publication where the flattening techniques are applied or described. Our findings exhibit a balanced distribution of flattening use cases between validation and code generation purposes. We also demonstrate that flattening techniques are generally not described thoroughly, for these are often but a minor step of a larger process. Finally, the validation of the flattening technique, although essential to gain confidence in the engulfing approach, is insufficiently addressed. This latter point is supported by preliminary experimentation indicating that only a small number hierarchy and parallelism levels may be supported.

The remainder of this paper is organised as follows. Section II presents our mapping study process. Section III presents the systematic map and discusses the results. Section IV describes our tool assessment and Section V covers threats to our empirical evaluation. Section VI wraps up with conclusions and future research directions. A companion webpage including all details of the mapping is available: https://staff.info.unamur.be/xde/mappingstudy/.

II. Systematic Mapping Process

The definition of our systematic mapping process is inspired by [3], [5], [6]. However, as suggested by Petersen et al. [3], the process presented in Figure 1 and detailed hereafter does not strictly follow the classical systematic mapping review process. It incorporates practices of systematic literature reviews methods: the depth of reading is not limited to the abstract of the publications but rather adapted according to the importance of flattening in the publication; a quality assessment phase (see Section III) has been added to evaluate the quality of the flattening description in the publications.

Phase 1: Research questions. The first phase is the definition of the research questions. They help delimiting the scope of the considered publications and allow to derive the search strings for publications exploration in phase 2. There

2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)
12th Workshop on Advances in Model Based Testing (A-MOST 2015)
978-1-4799-1885-0/15/$31.00 ©2015 IEEE

https://doi.org/10.1109/ICSTW.2015.7107408

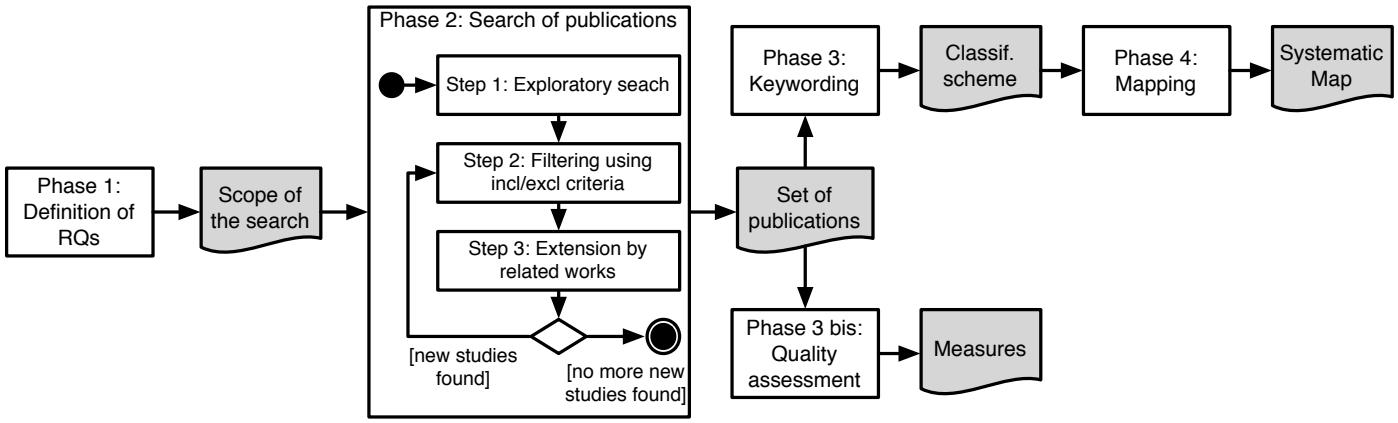

Fig. 1: Systematic mapping process

are two points covered in this mapping study: the first one examines flattening techniques suited to eliminate hierarchy and parallelism (orthogonality) from a state machine-like model while the second one looks at the application contexts of such transformations. To cover flattening techniques, 2 research questions with a practical perspective are defined: (*RQ1*) *What are the input and output models used in the different flattening approaches?* (*RQ2*) *Do the different approaches support hierarchy (composite states) and parallelism (orthogonal states) in the input model?* Flattening application context is covered by our last research question: (*RQ3*) *In which context are the different flattening approaches performed (e.g., code generation, test-case generation, semantic definition, etc.)?*

Phase 2: Search of publications. In the second phase of the process, we gather relevant publications used to build the systematic map. To that aim, we follow the strategy presented in Figure 1. First, we explore electronic databases and gather a raw set of publications. Next, we filter this raw set, and consequently obtain an initial set of relevant publications. From this initial set, we search for related (i.e. cited) work. The discovered papers are filtered and then added to the set of considered publications. We repeat the process until no new publication is added. To perform the exploratory search in the electronic databases, we defined search strings designed to answer our research questions and inspired from four publications known by the team and experts of the domain [7]–[10]. The considered databases with the different search strings are:

1) Google Scholar (http://scholar.google.be/):
   `(computer science) AND (intitle:( state AND (machine OR machines OR chart OR charts))) AND (flattening ) AND (orthogonal OR parallel)`
2) Science Direct (http://www.sciencedirect.com):
   `pub-date > 2004 and (("state machine" OR "state charts") AND (flat) AND (orthogonal OR parallel)) and TITLE(state)[All Sources(Computer Science)]`
3) Computer Science Bibliographies (http://liinwww.ira.uka.de/bibliography/):
   `+(flat flattening) +state +(diagram? chart?)`

The initial search string used for Google Scholar did not contain the `computer science` keywords which lead to a lot of irrelevant results, most of them related to chemistry. We initially restricted ourselves to the [2005-2012] period to gather approaches compatible with the current version of UML.[1] We found 167 publications in Google Scholar, 9 in Science Direct and 39 in Computer Science Bibliographies. The four publications known by the team and domain experts were contained in the 167 results returned by Google Scholar. This tends to indicate the relevancy of our search strings. In the second step, we filter the result according to the following inclusion and exclusion criteria:

Inclusion criteria. Books, articles, proceedings, technical reports and grey literature presenting a flattening technique with a hierarchical and / or orthogonal state machine or similar input (e.g., Harel Statecharts [1], etc.) and a flat state machine or assimilate as output (e.g., Finite State Machine (FSM), etc.) are included. We also consider the publications where the produced output is source code, as source code may be used for testing and verification as well.

Exclusion criteria. Literature only available in the form of presentation slides, publications where a flattening technique is only mentioned without details and publications citing a flattening technique described in another paper is excluded. In this last case, the cited papers are nonetheless considered according to the same inclusion and exclusion rules.

Once filtered, our set was extended in the third step by including papers cited in the publications. If the publication is focused on flattening, the references are picked up by screening the introduction, the background and related work parts. If not, only the "flattening related part", found by performing a word search in the documents, is considered for the references search. The considered regular expressions for the word search were: $flat.*$; $hierarch.*$; $orthogonal.*$. We repeat Steps 2 and 3 until our set of publications does not change.

[1]UML2 was released in 2005. Please note that only the *initial* scope of the publications is limited to the period 2005-2012. Our search process iteratively expands that period and eventually allows to consider "non UML" papers, e.g. [7], [10]–[13].

TABLE I: Research Focus Facet (RQ3)

Category	Description
Code generation	A model-driven approach generate or annotate source code from a flat state machine.
Model checking	A model to check is first flattened and the result is used as input of the model checker.
Formal semantics	The semantics of a state machine language is given as a transformation of which flattening is a step.
Model-based testing	Test-cases are generated from a flat state machine.
Flattening	Flattening is studied outside the scope of a specific application.
Example	Flattening illustrates the use of a particular transformation framework.

TABLE II: Input Model Facet (RQ1, RQ2)

Category	Description
UML state machine	State machines built according to (any version of) the UML standard.
Hierarchical Finite State Machine	Hierarchical models based on state machines (e.g., Harel statechart [1]).
Hierarchical Timed Automata	Hierarchical state machines enriched with time information.

TABLE III: Output Model Facet (RQ1, RQ2)

Category	Description
Flat UML state machine	Flat state machines based on any version of UML.
Source code	Code issued from a programming language or a textual specifications with a formal executable semantics.
Model checker specification	Any model checker specification, *e.g.*, UPPAAL automata [43] or Mealy machines [44].
Finite State Machine (FSM)	This facet regroups the publications where the flattening transformation produces a flat FSM which is not a UML state machine. For instance: EFSM, Harel statechart, Symbolic transition system.
Graph	Any kind of graph that other than finite state machine, *e.g.*, petri net or testing flow graph [11].

After an initial filtering, we obtained an initial set of 24 publications [7]–[30]. A first execution of Step 3 gave us 28 new papers. Among those, only 12 met the inclusion and exclusion criteria [31]–[42]. A second application of Step 3 gave us only one new publication, which did not match the inclusion and exclusion criteria. We eventually obtained a total of 36 publications.

Phase 3: Keywording and Mapping. The classification scheme follows a two-step keywording process inspired by that of Petersen *et al.* [3]. In the first step, each paper is screened and tagged with keywords representing its contribution. In the second step, the classification scheme is built by grouping and combining the different keywords in higher level categories. Contrary to [3], where the considered publications are focused on the subject of the mapping study, we also consider papers where hierarchical / orthogonal models flattening is not the main aspect of the publication but only a step in a more general process. To deal with such cases, we propose to: (1) read only sections (adaptive reading depth [3]) of publications where the model flattening aspect is explained and (2) guide the keywording process by our research questions: a) The purpose of the research (*RQ3*). b) The input model of the transformation (*RQ1*). c) The output model of the transformation (*RQ1*). d) Does the transformation support hierarchy / orthogonality in the input model? (*RQ2*). e) The implementation of the transformation (*RQ3*). In order to reduce bias, the first step of the keywording process has been done in parallel by two reviewers. The reviewers associate keywords with each publication, compare their results and discuss the differences. If they cannot agree on a given paper, a third reviewer solved the conflict. In our case, this happened for two papers: [27], [40].

Next, the classification scheme is built by clustering the keywords into different categories. Similar categories are grouped to form what is called a facet. This is an iterative process where the classification scheme is enriched with each newly considered publication. In our case, four facets compose the classification scheme. The first facet (see Table I) is concerned with the focus of the research described in the publication. This characterizes the broader context in which the flattening transformation is used. The second and third facets (see Tables II and III, respectively) describe the formalisms of the input and output models (respectively) employed by the flattening techniques described in the different publications. The last facet (see Table IV) classifies publications according to their type. These types range from problem-oriented papers (opinion, philosophical paper) to solution-oriented papers at various stages of their maturity.

Once the classification scheme is defined, all the publications are classified. Despite our inclusion / exclusion criteria, we still found publications which, after complete review, were irrelevant with regards to our research questions: Brajnik [17] presents the flattening proposed by Wasowski [10]; Briand *et al.* [18] briefly discuss flattening in the related work part; Masiero and Maldonado [22] present a way to produce a reachability tree for hierarchical state machines which can not be considered as a flattening; in [23] Posse preserves the hierarchical aspect of state machine in its target model; Zoubeyr *et al.* [30] do not describe any flattening technique; Engels *et al.* [38] describe a flattening of UML class hierarchy. All those publications match the exclusion criteria (flattening technique is only mentioned or comes from another paper) but were not detected earlier because of seemingly "too broad" regular expressions. Irrelevant papers have been removed. Our final selection consists of 30 classified publications.

Phase 3 bis: Quality Assessment. In parallel with Phase 3, we propose, as suggested by da Mota Silveira Neto *et al.* [5] to assess the quality of the selected publications using two groups of quality criteria. Our evaluation does not focus on the quality of the transformations themselves but rather on the quality of its description in the publications. The first group evaluates the usability of the flattening technique in the publication: G 1.1 Is there a tool implementation? G 1.2 Is there a small example? G 1.3 Is there a more significant case study (even if not fully detailed)? G 1.4 Are the input and output models described? G 1.5 Does the publication present the limitations of the transformation?

The second group evaluates the degree of generality of the flattening process in the publication: G 2.1 Are there guidelines for the transformation separated from the example of the transformation (if any)? G 2.2 Does it detail the transformation process for all the constructs of the input model? G 2.3 Does the flattening technique support hierarchy? G 2.4 Does the

TABLE IV: Research Type Facet (RQ3) [3]

Category	Description
Validation Research	Techniques investigated are novel and have not yet been implemented in practice. Techniques used are for example experiments, i.e., work done in the lab.
Evaluation Research	Techniques are implemented in practice and an evaluation of the techniques is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes identifying problems in industry.
Solution Proposal	A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation.
Philosophical Papers	These papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework.
Opinion Papers	These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodologies.
Experience Papers	Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author.

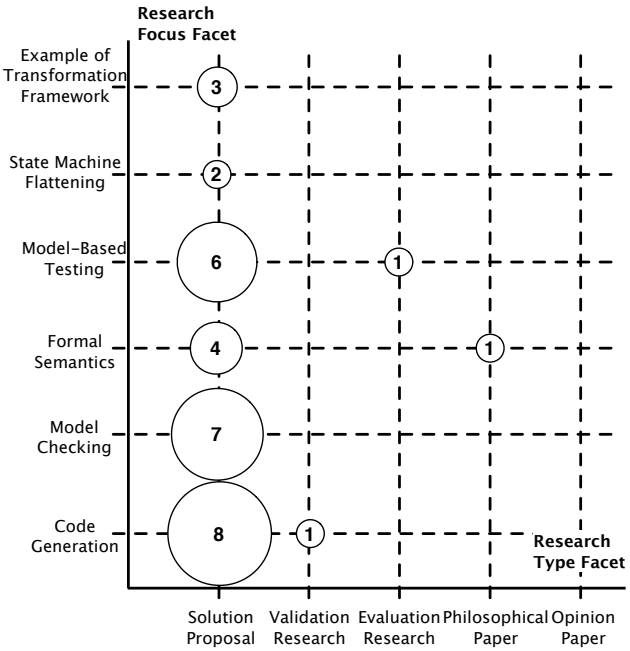

Fig. 2: Systematic Map: Research type and focus facets

flattening technique support orthogonality? A seemingly more objective metric is the ratio of text lines dedicated to the flattening process. Yet, it is rather cumbersome to perform and may be irrelevant since (1) longer descriptions are not necessarily more precise and complete, and (2) the exact value of this ratio may vary upon the reviewers.

Results of quality assessment are presented in Table V. It turns out that the huge majority (all but [34]) of the publications agree to define "flattening" as the "removal of hierarchy" in a state machine. In 97% of the cases (Q.2.3), the input model supports hierarchy of states, and 70% also supports orthogonality in sub-states. The only publication present in the mapping without supporting hierarchical states is [34]. We believe this publication should have been discarded by the protocol. Baresi and Pezzé [34] define the semantics of state machines as high-level Petri-nets, but the notion of hierarchy is applied to classes (although not as explicitly as in [38]) and not states. Yet, the method may be applicable on hierarchical state machines (see [21]).

Only 60% of the publications thoroughly explain which constructs of the input model are supported (Q.2.2). These publications include: three technical reports (T. rep.) [8], [14], [19], one PhD thesis [15] and one book [16]. Those kinds of publications typically allow for more space to provide details than articles (Art.), book sections (B. sect.) or proceedings (Proc.). Two of the three publications belonging to the "Example of transformation framework" facet, [9], [40], are also included. The third publication [31] does not explain the complete transformation in detail but rather focuses on its performance. In most other cases [7], [11]–[13], [20], [24], [29], [32], [33], [40], the input model is a simplified version of UML state machine (except for [33] which uses Harel statecharts as input) where most of the pseudo-states have not been taken into account. In all those cases, the limitations of the transformation are presented in the publication (Q.1.5: 67%). The only flattening techniques which takes history pseudo-state into account is the one presented by Wasowski et al. [10], [27]. Finally, the transformation proposed by David and Möller [19] uses Hierarchical Timed Automata as input. This formalism has a well-defined semantics, contrary to UML state machines.

We observe that only a very low percentage of techniques are validated on real or consequent case studies (Q.1.3: 27%). It is not surprising since most of the publications are solution proposals. Only half of the publications have a tool implementation (Q.1.1). 90% of the publications illustrate the transformation with examples (Q.1.2), 90% describe the input and output models (Q.1.4), and 80% provide more detailed guidelines. Only [26] and [37] give no example nor guideline. The former describes very briefly the transformation in terms of input and output models. The latter presents the specifications and an overview of a model-checking tool without discussing the flattening transformation or the input model.

III. Phase 4: Mapping

The complete mapping is not presented here due to space constraints. Figures 2 and 3 present a view of the mapping in the form of a bubble plot. The numbers in the bubbles represents the numbers of studies belonging to a particular combination of facets. In Figure 2, the number of studies is equal to 33. This is due to the classification of [15], [19] and [20]. [15] is a PhD thesis classified as a validation research and a solution proposal in the research type facet. In [19] and [20] David *et al.* uses flattening in order to generate code for a model checker, the reviewers agree to classify those two studies in both code generation and model checking in the research focus facets.

Research Type (RQ3). Regarding the type of contributions where flattening considerations appear, the vast majority of them (93%) are solution papers (see Figure 2). This is to be expected since flattening is a transformation used to bridge high-level models with existing lower level analysis

TABLE V: Quality assessment results

Study ref.*	G1.1	G1.2	G1.3	G1.4	G1.5	G2.1	G2.2	G2.3	G2.4	Type
Initial set										
Auer [14]	✓	✓		✓	✓	✓	✓	✓	✓	T. rep.
Badreddin [15]	✓	✓	✓	✓	✓	✓	✓	✓	✓	Thesis
Binder [16]		✓		✓	✓	✓	✓	✓	✓	Book
David [19]	✓	✓	✓	✓	✓	✓	✓	✓	✓	T. rep.
David [20]	✓	✓	✓	✓	✓	✓	✓	✓	✓	Art.
Gogolla [7]		✓		✓	✓	✓	✓	✓		Art.
Holt [8]	✓	✓		✓	✓	✓	✓	✓	✓	T. rep.
Ipate [21]		✓		✓				✓	✓	Art.
Kalnins [9]	✓	✓		✓	✓	✓	✓	✓	✓	Art.
Kansomkeat [11]		✓		✓	✓	✓	✓	✓		Proc.
Kim [12]		✓		✓	✓	✓	✓	✓	✓	Art.
Kuske [13]		✓		✓	✓	✓	✓	✓	✓	Art.
Sacha [24]		✓	✓	✓	✓	✓	✓	✓		Art.
Schattkowsky [25]		✓		✓		✓		✓	✓	Art.
Schwarzl [26]				✓				✓		Proc.
Wasowski [10]	✓	✓	✓	✓	✓	✓	✓	✓	✓	Art.
Wasowski [27]		✓		✓	✓	✓	✓	✓	✓	Art.
Weißleder [28]	✓	✓						✓		Proc.
Yao [29]	✓	✓	✓	✓	✓	✓	✓	✓	✓	Proc.
Added after iteration 1										
Agrawal [31]	✓	✓		✓		✓		✓	✓	T. rep.
Ali [32]		✓		✓	✓	✓	✓	✓	✓	Proc.
Andrea [33]		✓		✓	✓	✓	✓	✓	✓	Proc.
Baresi [34]		✓	✓	✓		✓				B. sect.
Bjorklund [35]	✓	✓		✓				✓	✓	Proc.
Bond [36]	✓	✓						✓		Proc.
Diethers [37]	✓							✓		B. sect.
Hong [39]	✓	✓		✓	✓	✓		✓	✓	Art.
Minas [40]		✓	✓	✓	✓	✓	✓	✓	✓	Art.
Riebisch [41]	✓			✓		✓		✓		Proc.
Roubtsova [42]		✓		✓	✓	✓		✓	✓	Proc.
Ratio of ✓	**50%**	**90%**	**27%**	**90%**	**67%**	**80%**	**60%**	**97%**	**70%**	

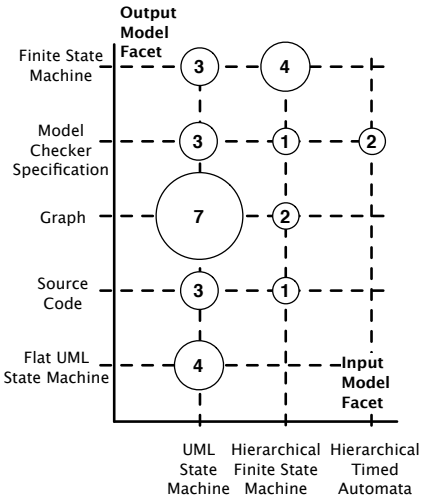

Fig. 3: Systematic Map: Input and output facets

tools and execution frameworks. We also note the poor level of validation and evaluation of the flattening algorithms (only 2 papers belong to the evaluation and validation research facets). Among these two publications, only [28] evaluated the effects of flattening on test case generation in practice. Finally, one paper [27] considers a formal framework to discuss flattening algorithms complexity.

Research Focus (RQ3). The research focus facet illustrates a balanced distribution of the applications of flattening. The most common application of flattening is code generation (27%). However this has to be mitigated by the fact that two publications [19], [20] are producing code to be used with a model checker (UPPAAL). Other kinds of generated code are dedicated to the synthesis of embedded systems [10], [24], [35] or are instances of general purpose languages like JAVA [32]. Model checking uses are related to consistency management [26], [29], [37], [42] or IP telephony [36]. Model checking exploits the fact that flattening is also a way to provide a formal semantics to hierarchical state machines [7], [13], [27], [33]. Three of these publications focus on UML state machines to make them analyzable. Model-based testing also extensively uses flattening approaches (23% of the publications). Unsurprisingly, most of the applications are centred on test case generation and selection [11], [12], [21], [28], [39], [41]. Rather than generating test cases over a flattened state machine, Ipate [21] and Binder [16] propose to refine test cases gradually as states are decomposed. They argue that this incremental approach better copes with complexity. Riebisch *et al.* [41] use state diagrams to refine UML use cases and subsequently generate tests at the system level. As for the other approaches, the generation algorithm requires a flat state machine. Two publications ([12], [39]) flatten Harel's statecharts and UML state machines (respectively) in order to generate flow graphs on which test-case generation and selection techniques are applied. Kansomkeat *et al.* [11] flatten UML state machines to testing flow graphs in order to generate test-cases. Weißleder [28] flatten UML 2 state machines and uses coverage criteria to select and generate test-cases. The two last categories (covering 17% of the publications) concern the flattening transformation by itself. Holt *et al.* [8] describe in details a flattening algorithm implemented as a model transformation embedded in an Eclipse plug-in. As opposed to other publications [13], [25], [33] based on graph transformations, the algorithm is

TABLE VI: Results: Average execution time

Depth	UMPLE	SM2LIME	SCOPE
0	0,306 sec.	0,038 sec.	$<$ 0,001 sec.
1	0,384 sec.	0,040 sec.	0,012 sec.
2	0,510 sec.	0,050 sec.	0,012 sec.
3	Error	2,726 sec.	Error
4	Error	$>$ 24 hrs	Error

given in an imperative manner. Finally, state machine flattening transformations are sometimes given as illustrative examples of model transformation frameworks [9], [31], [40].

Input and Output Facets (RQ1, RQ2). UML models are the most common input to flattening algorithms (67%). There are, however, disparities in the supported UML constructs. The output of a flattening algorithm mainly depends on the goal for which the techniques is used. Graphs are preferred for providing formal semantics whereas verification-related work generally provides specifications for a model checker. If we match inputs with outputs, we infer that flattening is essentially an *exogenous* transformation (*i.e.* where the target language is different from the source language): UML state machines are both input and output in only 20% of the publications.

IV. Preliminary Tool Assessement

Our mapping study revealed a certain interest of the community for the flattening problem. A significant number of solutions have also been provided. Yet, tools are available in only 50% of the publications and validation remains rare. Practical questions concerning existing tool support naturally arise. Thus, we decided to conduct an additional assessment of available tool support in the form of experiments. In particular we focused on one particular question: *How do the proposed techniques scale to models of increasing complexity?*

Selection. Amongst the 15 publications that include an implementation [8]–[10], [14], [15], [19], [20], [28], [29], [31], [35]–[37], [39], [41] only 5 tools could be found on the Internet [10], [14], [15], [28], [31]. We picked the three of them that have a command-line interface: SCOPE [10], UMPLE [15] and SM2LIME [14]. To evaluate their performance, we fed them with a state machine example that we successively extend with an increasing number of composite and orthogonal states. Since we are interested in reusable flattening techniques and we want our experiments to be reproducible, we considered only publicly available tools and did not contact the authors to obtain their implementation. Moreover, we are aware that our evaluation does not cover every existing tool as some are not presented in an elicited publication.

Experiment Design. Input models of varying complexity were automatically generated as follows. We started from a simple state machine sm_0 as base model with an initial state i, a simple state s and a final state f and two transitions: one from i to s and one from s to f triggered by an event with zero parameters. This machine with no composite state has a depth equal to 0. To produce state machine sm_k with a depth equal to k ($k \in \{0, 1, 2, \ldots, 10\}$), we replaced the simple states in sm_{k-1} by a composite state with two orthogonal regions containing each one a sm_0 state machine. We run each tool on the ten input models and measure their execution time using the Unix time (`/usr/bin/time`) command available on a Linux machine (kernel version: #61-Ubuntu SMP Tue Feb 19 12:39:51 UTC 2013) with a Intel Core i3 (3.10GHz) processor and 4GB of memory. To minimize effects due to other running processes, we repeated each experiment five times.

Results & Discussion. Table VI presents the average execution time of each tool. None of the three selected tools could achieve more than 3 levels of depth: *UMPLE* exits with a syntax error although sm_3 is generated using the same procedure as sm_2; *SCOPE* exits with a memory corruption error at sm_3; *SM2LIME* could process a sm_3 state machine but with an execution time jumping from 0,050 to 2,726 and has an (extrapolated) execution time greater than 24 hours for a sm_4 input model. Although the input models look simple, they become increasingly challenging due to an exponential blow-up in the numbers of parallel regions and interleaving transitions. Moreover, the structure of our models impedes the use of various optimization (*e.g.* eliminating superfluous state/transitions using guard analysis [10]) and thus yield a sharper growth in complexity. Thus, these models are not intended to reflect any real system; they are meant to measure the scalability of the proposed tools. Additionally, they are agnostic of semantic variations of the different formalisms [45], [46]. This allows for fair comparisons between the tools.

State explosion did not allow for fine-grained trend analysis as models grow. However, our experiments confirm conclusions drawn on the mapping study regarding limited availability (overall only 33% of the tools are freely available) and suggest that hierarchy and parallelism threaten scalability. This further motivates the need for new efficient techniques.

V. Threats to validity

Publication bias. We cannot guarantee that all relevant publications were selected, especially since the state machine flattening is rarely the main focus of the publications but rather a way to achieve a more general purpose. We tried to mitigate this threat by adopting an approach were the set of publications is built iteratively by including cited papers. The publication dates of the papers added at each step of publications search (Phase 2) shows a good coverage for a period from 1994 to 2012: in the initial set the oldest publication ([22]) was published in 1994 and the most recent ([15]) was published in 2012; the publications added in iteration 1 were less recent (from 1994 [33] to 2008 [40]) and the publication found during iteration 2 (and excluded from the set of publications) was published in 1996 ([47]). Moreover, the selected papers originate from different research areas, thus indicating that our selection procedure covers a large scope of publication. Finally, the cited documents of rejected publications were still included in the set at step 3 of the "search of publications" phase.

Research strings. The search strings used for database mining may have many synonyms. Relevant publications may thus remain undetected. Still, the used strings allowed us to successfully detect the four initial papers known by the domain experts. As for publication bias, the distribution of the publication dates from 1994 to 2012 shows that the initial publication period is no major threat to completeness.

Keywording. As the considered publications are not all focused on flattening, the keywording process may be influenced by the reviewer. To avoid bias, the keywording

process is performed in parallel by two different readers. Once the keywords have been associated with the publications, the readers compare their results and discuss the differences between associated keywords. If conflicts between the associated keywords remain, a third party acts as an arbitrator.

Quality assessment. As for the keywording process, the point of view of the reviewer may influence the answers to the different questions. To overcome this as much as possible, only yes/no answers are allowed. Since the quality assessment was performed in parallel with the keywording phase, the two reviewers have assessed the quality of the different publications. Again, divergences were solved by a third party.

Tool Selection. The rationale behind tool selection was to assess whether tools mentioned in the publications were publicly available and ready for practical use. While our answer is negative to these questions, efficient tools may have been missed because of our focus on scientific publications. This threat can be mitigated by the fact that proper documentation is necessary to understand the tool's input model formalism and thus generate models for experimentation. However, conducting a wider assessment on a larger set of tools is part of our research agenda.

Model Complexity. We created and systematically extended challenging models. Such an approach is relevant to compare tools on a fair basis. To the best of our knowledge, there exists no survey about the size and complexity of state machines designed in industry. It is thus possible that such a high level of complexity never occurs in practice.

VI. Conclusion

Due to their compactness and formal semantics, state machines are a powerful means of modelling, verifying and validating the behaviour of complex systems. However, abstraction mechanisms such as composite and parallel states impede the use of automated analysis and generation techniques, often requiring flat structures. Flattening is called to play a crucial role in bridging abstract models with analysable and executable ones. Recognising the lack of overall cartographies of flattening approaches, this systematic mapping study is a first step in this direction. In particular, we outlined a balanced status were flattening is used equally for model-based validation (testing, verification) and code generation. Flattening also barely appears to be an object of interest in itself but rather a step towards a more general objective. This has impacts on the quality of the description of flattening algorithms. First, precise constructs supported by the flattening transformation are not always provided, making the applicability of a given technique to a specific context difficult to evaluate. Second, the validity of the flattening transformation is barely addressed, which is necessary to gain confidence in the quality of the bridge. Mapping study conclusions are supported by our preliminary assessment of flattening tools that exhibited reliability and scalability issues on small but challenging models.

In the future, we would like to provide a complete (including syntax and semantics concerns) taxonomy of flattening approaches. This will enable the design of generic flattening techniques and tools. We will also offer a sound evaluation framework to compare flattening techniques and thus will help understanding in which situation(s) a given flattening approach is the most appropriate. These are mandatory steps towards reliable, end-to-end, model-based behavioural development.

References

[1] D. Harel, "Statecharts: a visual formalism for complex systems," *SCP*, vol. 8, no. 3, pp. 231–274, Jun. 1987.

[2] X. Devroey, M. Cordy, G. Perrouin, E.-Y. Kang, P.-Y. Schobbens, P. Heymans, A. Legay, and B. Baudry, "A Vision for Behavioural Model-Driven Validation of Software Product Lines," in *ISoLA '12*, 2012, pp. 208–222.

[3] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *EASE*, Bari, Italy, 2008, pp. 71–80.

[4] B. Kitchenham, "Guidelines for performing Systematic Literature Reviews in Software Engineering," EBSE, Tech. Rep. EBSE-2007-01, 2007.

[5] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing," *IST*, vol. 53, no. 5, pp. 407–423, 2011.

[6] E. Engström and P. Runeson, "Software product line testing-a systematic mapping study," *IST*, vol. 53, no. 1, pp. 2–13, 2010.

[7] M. Gogolla and F. Parisi Presicce, "State diagrams in UML: A formal semantics using graph transformations," in *PSMT*, 1998.

[8] N. E. Holt, E. Arisholm, and L. C. Briand, "An Eclipse Plug-in for the Flattening of Concurrency and Hierarchy in UML State," Simula Research Laboratory, Norway, Tech. Rep. 2009-06, 2010.

[9] A. Kalnins, J. Barzdins, and E. Celms, "Model transformation language MOLA," *Model Driven Architecture*, vol. LNCS 3599, pp. 62–76, 2005.

[10] A. Wasowski, "Flattening statecharts without explosions," *ACM Sigplan Notices*, vol. 39, no. 7, pp. 257–266, 2004.

[11] S. Kansomkeat and W. Rivepiboon, "Automated-Generating Test Case Using UML Statechart Diagrams," in *SAICSIT '03*, South Africa, 2003, pp. 296–300.

[12] Y. G. Kim, H. S. Hong, D. H. Bae, and S. D. Cha, "Test cases generation from UML state diagrams," *IEE Proceedings - Software*, vol. 146, no. 4, p. 187, 1999.

[13] S. Kuske, "A Formal Semantics of UML State Machines Based on Structured Graph Transformation," in *UML '01*, ser. LNCS 2185. Springer-Verlag, 2001, pp. 241–256.

[14] E. Auer and I. Porres, "SM2LIME : A Translation Tool From UML State Machines to LIME Specifications," IT Dpt. Abo Akademi University, Tech. Rep., 2009.

[15] O. Badreddin, "A Manifestation of Model-Code Duality : Facilitating the Representation of State Machines in the Umple Model-Oriented Programming Language," Ph.D. dissertation, University of Ottawa, 2012.

[16] R. V. Binder, *Testing object-oriented systems: models, patterns, and tools*. USA: Addison-Wesley, 1999.

[17] G. Brajnik, "Using UML State Machines for Interaction Design and Usability Evaluation: An Extensive Literature Review," Web Ergonomics Lab, School of Computer Science, University of Manchester, UK, Tech. Rep. September, 2011.

[18] L. Briand, Y. Labiche, and Q. Lin, "Improving Statechart Testing Criteria Using Data Flow Information," *ISSRE*, pp. 95–104, 2005.

[19] A. David and M. O. Möller, "From HUppaal to Uppaal : A translation from hierarchical timed automata to flat timed automata," BRICS, University of Aarhus, Denmark, Tech. Rep. March, 2001.

[20] A. David, M. O. Möller, and W. Yi, "Formal Verification of UML Statecharts with Real-Time Extensions," in *FASE '02*, 2002, pp. 218–232.

[21] F. Ipate, "Test Selection for Hierarchical and Communicating Finite State Machines," *The Computer Journal*, vol. 52, no. 3, pp. 334–347, May 2008.

[22] P. Masiero, J. Maldonado, and I. Boaventura, "A reachability tree for statecharts and analysis of some properties," *IST*, vol. 36, no. 10, pp. 615 – 624, 1994.

[23] E. Posse, "Mapping UML-RT State Machines to kiltera," Applied Formal Methods, Group School of Computing, Queen's University, Tech. Rep. 2010-569, 2010.

[24] K. Sacha, "Translatable Finite State Time Machine," in *SDL Forum*. Paris, France: Springer-Verlag, 2007, pp. 117–132.

[25] T. Schattkowsky and W. Müller, "Transformation of UML State Machines for Direct Execution," in *VLHCC*. IEEE Computer Society, 2005, pp. 117–124.

[26] C. Schwarzl and B. Peischl, "Static and Dynamic Consistency Analysis of UML State Chart Models," in *MODELS*. Oslo, Norway: Springer-Verlag, 2010, pp. 151–165.

[27] A. Wasowski, "On Succinctness of Hierarchical State Diagrams in Absence of Message Passing," *ENTCS*, vol. 115, pp. 89–97, 2005.

[28] S. Weißleder, "Influencing Factors in Model-Based Testing with UML State Machines : Report on an Industrial Cooperation," in *MODELS*. Denver, CO: Springer-Verlag, 2009, pp. 211–225.

[29] S. Yao and S. Shatz, "Consistency Checking of UML Dynamic Models Based on Petri Net Techniques," in *CIC*. Washington, DC, USA: IEEE, Nov. 2006, pp. 289–297.

[30] F. Zoubeyr, A. Tari, and A. M. Ouksel, "Backward validation of communicating complex state machines in web services environments," *Distributed and Parallel Databases*, vol. 27, no. 3, pp. 255–270, Mar. 2010.

[31] A. Agrawal, G. Karsai, and F. Shi, "Graph Transformations on Domain-Specific Models," ISIS, Vanderbilt University, USA, Tech. Rep. Mic, 2003.

[32] J. Ali and J. Tanaka, "Converting Statecharts into Java Code," in *IDPT*, Dallas,Texas, 1999, p. 42.

[33] M.-S. Andrea and A. Peron, "Semantics of full statecharts based on graph rewriting," in *Graph Transformations in Computer Science*. Springer, Berlin, 1994, pp. 265–279.

[34] L. Baresi and M. Pezzé, "On Formalizing UML with High-Level Petri Nets," in *Concurrent object-oriented programming and petri nets*. Springer, 2001, pp. 276–304.

[35] D. Björklund, J. Lilius, and I. Porres, "Towards Efficient Code Synthesis from Statecharts," in *pUML-Group@UML '01*, 2001, pp. 29–41.

[36] G. W. Bond, F. Ivancic, N. Klarlund, and R. Trefler, "ECLIPSE Feature Logic Analysis," in *2nd IP-Telephony Workshop*, New York City, USA, 2001, pp. 100–107.

[37] K. Diethers and M. Huhn, "Vooduu : Verification of Object-Oriented Designs Using UPPAAL," in *TACAS '04*, ser. LNCS 2988. Springer, 2004, pp. 139–143.

[38] G. Engels, J. H. Hausmann, R. Heckel, and S. Sauer, "Dynamic Meta Modeling : A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML," in *UML*. York, UK: Springer-Verlag, 2000, pp. 323–337.

[39] H. S. Hong, Y. G. Kim, S. D. Cha, D. H. Bae, and H. Ural, "A test sequence selection method for statecharts," *STVR*, vol. 10, no. 4, pp. 203–227, Dec. 2000.

[40] M. Minas and B. Hoffmann, "An Example of Cloning Graph Transformation Rules for Programming," *ENTCS*, vol. 211, pp. 241–250, Apr. 2008.

[41] M. Riebisch, I. Philippow, and M. Götze, "UML-Based Statistical Test Case Generation," in *NODe '02*. Erfurt, Germany: Springer-Verlag, 2003, pp. 394–411.

[42] E. E. Roubtsova, J. van Katwijk, R. C. M. de Rooij, and H. Toetenel, "Transformation of UML Specification to XTG," in *PSI '02*, 2001, pp. 247–254.

[43] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal - a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. LNCS, vol. 1066, 1995, pp. 232–243.

[44] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, pp. 1045–1079, 1955.

[45] A. Taleghani and J. Atlee, "Semantic variations among uml statemachines," in *MODELS'06*, ser. LNCS 4199. Springer, 2006, pp. 245–259.

[46] M. Crane and J. Dingel, "Uml vs. classical vs. RHAPSODY statecharts: Not all models are created equal," in *MDE Languages and Systems*. Springer, 2005, pp. 97–112.

[47] D. Harel and E. Gery, "Executable object modeling with statecharts," in *ICSE*. Berlin, Germany: IEEE, 1996, pp. 246–257.