Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

Sismic—A Python library for statechart execution and testing

Alexandre Decan [*], Tom Mens

Software Engineering Lab, University of Mons, Belgium

A R T I C L E I N F O

Article history:
Received 15 May 2019
Received in revised form 9 March 2020
Accepted 14 September 2020

Keywords:
Statechart
Executable modelling
Behaviour-driven development
Design by contract
Runtime verification

A B S T R A C T

Statecharts are a well-known visual modelling language for representing the executable behaviour of complex reactive event-based systems. The essential complexity of statechart models solicits the need for advanced model testing and validation techniques, such as test-driven development, behaviour-driven development, design by contract, and property statecharts for monitoring of violations of behavioural properties during statechart execution. *Sismic* is an open-source Python library providing a tool suite to define, simulate, execute and test statecharts with all of the aforementioned techniques.

Current code version
Code metadata

Current code version	1.4.1 (commit d61e748)
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_181
Code Ocean compute capsule	Not applicable
Legal Code Licence	GNU Lesser General Public Licence version 3.0 (LGPLv3)
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python 3.5+; ruamel.yaml, schema, behave, typing
If available Link to developer documentation/manual	https://sismic.readthedocs.io
Support email for questions	alexandre.decan@lexpage.net

Current executable software version
Software metadata

Current software version	1.4.1
Permanent link to executables of this version	https://github.com/AlexandreDecan/sismic
Legal Software Licence	GNU Lesser General Public Licence version 3.0 (LGPLv3)
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
Installation requirements & dependencies	Python 3.5+; ruamel.yaml, schema, behave, typing
If available, link to user manual–if formally published include a reference to the publication in the reference list	https://sismic.readthedocs.io
Support email for questions	alexandre.decan@lexpage.net

1. Motivation and significance

Statecharts are a visual executable modelling language introduced by David Harel [1] as an extension of hierarchical finite state machines with characteristics of both Mealy and Moore automata. Statecharts are part of the UML standard and constitute a popular notation for representing the executable behaviour of complex reactive event-based systems. They are frequently used in industry for the development of real-time systems and embedded systems, relying on commercial tools such as IBM Rational *Rhapsody*, The Mathworks *Stateflow*, itemis *Yakindu Statechart Tools*, IAR Systems *visualSTATE*, and QuantumLeaps *QM*. Most of these tools support visualisation, modification and simulation of statecharts, as well as code generation from statechart

[*] Corresponding author.
E-mail addresses: alexandre.decan@umons.ac.be (A. Decan), tom.mens@umons.ac.be (T. Mens).

https://doi.org/10.1016/j.softx.2020.100590

models. The more advanced tools also provide support for model debugging and model verification.

A wide variety of testing techniques and associated tools is available for developing source code in programming languages. These techniques include *test-driven development (TDD)* [2], *behaviour-driven development (BDD)* [3], and *design by contract (DbC)* [4]. These techniques have proven their usefulness for "classical" programming languages. Since the statechart formalism is Turing-complete [5], such techniques can also be beneficial during the development of executable statecharts.

For this purpose, we have developed *Sismic* (a recursive acronym for *Sismic Interactive Statechart Model Interpreter and Checker*), a modular Python library for executing, testing and validating executable statecharts based on the techniques of TDD, BDD, DbC, and property statecharts that allow to monitor for violations of behavioural properties during statechart execution. *Sismic* targets both researchers and practitioners interested in exploring and putting these techniques into practice in their software development projects. *Sismic* provides a flexible API to facilitate its use in regular Python code or to facilitate its extension by other researchers. *Sismic* has been validated through a controlled user study in [6] and is used by multiple companies.

2. Software description

Sismic is a statechart library for Python (version 3.5 or higher) providing a set of tools to define, execute and test statecharts. The library is distributed through the Python Package Index.[1] Its source code is available on GitHub[2] under the open-source licence LGPLv3. *Sismic* is extensively documented on sismic.readthedocs.ioand comes with an extensive test suite with high code coverage. The tests, examples and code fragments of the documentation are automatically executed as part of a continuous integration process.

2.1. Software functionalities

Sismic provides an easy way to define and import statecharts, based on the human-friendly YAML markup language. Visualisation of these statecharts is realised through an interface with PlantUML.

Sismic's statechart interpreter offers a discrete, step-by-step, observable simulation engine, supporting the main statechart concepts. The default interpreter uses an inner-first/source-state and run-to-completion semantics (a.k.a. big step semantics [7]), but it can be tuned to other semantics by subclassing this interpreter. To support timed events, the interpreter comes with controllable simulation clock that supports both real and simulated time. Statechart actions and guards can be expressed using regular Python code as the action language, but *Sismic* can be extended to support other action languages as well. *Sismic* also provides support for communication between statecharts, as well as for allowing regular Python code to be called from within statecharts and vice versa.

With respect to statechart testing, *Sismic* supports regular unit testing of statecharts; a DbC approach to specify invariants, pre- and postconditions on states and transitions [4]; BDD [3]; and runtime checking of behavioural properties that are expressed as statecharts themselves.

[1] pypi.org/project/sismic.
[2] github.com/AlexandreDecan/sismic.

2.2. Software architecture

Sismic provides a modular, easily extensible architecture, summarised in Fig. 1. Experienced Python developers may choose to directly create and manipulate statecharts through the sismic.model API. In practice, it is more convenient to create statecharts using either a text-based markup editor or an external visual editor, and import these models through the sismic.io API. This API has support for exporting statecharts to PlantUML (plantuml.com) in order to visualise them by means of automatic layout features. This is how Fig. 2 has been generated.

Sismic's main component is the sismic.interpreter module. In order to execute a statechart model, an interpreter must be instantiated. This interpreter relies on an *action code evaluator* to execute any code contained in the actions or guards of the statechart specification. Module sismic.code provides such an evaluator to express action code using regular Python.

The interpreter supports run-time monitoring of contract violations. Contracts are specified directly as part of the statechart description using the language supported by the action code evaluator, and can use a range of predefined predicates. The interpreter also provides built-in support for runtime monitoring properties expressed as statecharts. These properties are expressed as statecharts that express functional properties of the intended behaviour in terms of the events that are consumed or sent, or in terms of the states that are entered or exited by a statechart being monitored. Finally, module sismic.bdd provides support for BDD by providing a flexible way to define, map and execute *BDD* scenarios.

3. Illustrative example

To illustrate the use of *Sismic*, consider the example of a simplified microwave controller in Fig. 2, visually rendered with PlantUML. The code fragment below shows part of the YAML description of this statechart:

```
statechart:
  name: Simple microwave controller
  root state:
    name: microwave controller
    initial: door closed
    on entry: timer = 0
    states:
      - name: door opened
        on entry: send('lamp_on')
        on exit: send('lamp_off')
        transitions:
          - event: door_closed
            target: door closed
```

This statechart can be executed through API calls to *Sismic*'s interpreter:

```
from sismic.io import import_from_yaml
from sismic.interpreter import Interpreter
statechart = import_from_yaml(filepath='microwave.yaml')
interpreter = Interpreter(statechart)
interpreter.queue('timer_inc', 'cooking_start')
interpreter.execute()
assert 'cooking' in interpreter.configuration
```

Contracts are defined as part of the statechart. At runtime, the interpreter will verify the conditions specified by the contracts. The example below shows a contract for the "cooking" state of the microwave:

```
contract:
  - before: timer > 0
  - after: received('door_opened') or timer == 0
```

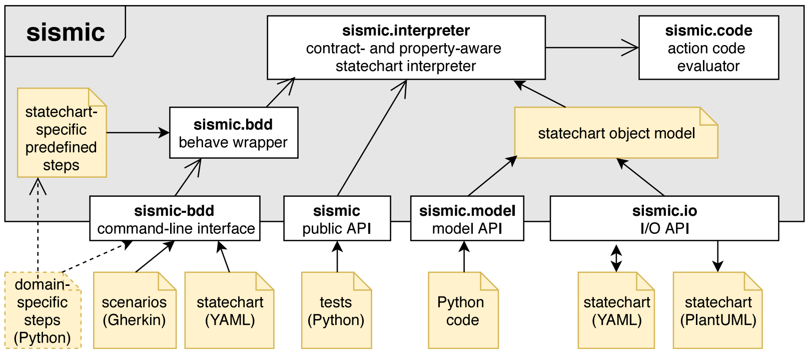

Fig. 1. High-level architectural overview of *Sismic*.

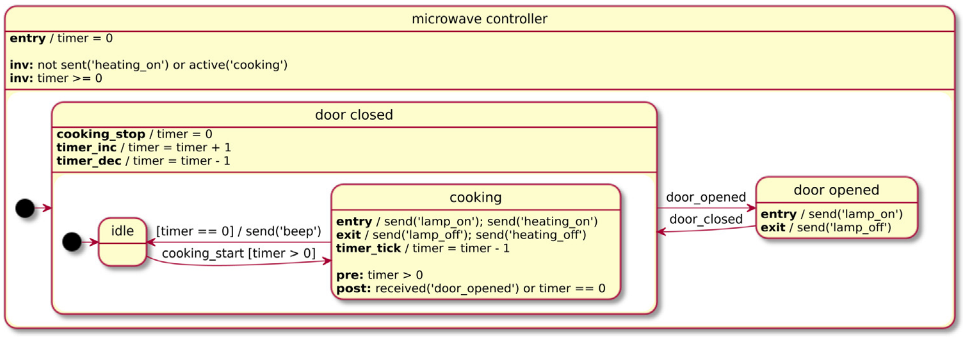
microwave controller
entry / timer = 0
inv: not sent('heating_on') or active('cooking')
inv: timer >= 0
door closed
cooking_stop / timer = 0
timer_inc / timer = timer + 1
timer_dec / timer = timer - 1
idle
[timer == 0] / send('beep')
cooking_start [timer > 0]
cooking
entry / send('lamp_on'); send('heating_on')
exit / send('lamp_off'); send('heating_off')
timer_tick / timer = timer - 1
pre: timer > 0
post: received('door_opened') or timer == 0
door_opened
door_closed
door opened
entry / send('lamp_on')
exit / send('lamp_off')

Fig. 2. Visual representation of a simplified microwave controller.

Sismic supports BDD to express scenarios or test cases by domain experts in natural language. This allows designers to focus on the purpose of the model rather than the technical details. Given an appropriate mapping, the following scenario can be executed and checked by *Sismic* using the command-line utility `sismic-bdd`:

```
Scenario: Heating is on while cooking
  Given I open the door
  And I place an item in the oven
  And I close the door
  And I press increase timer button 5 times
  When I press start button
  Then heating turns on
```

4. Impact

We have carried out a controlled user study to evaluate the feasibility, usefulness and adequacy of using *Sismic* for the purpose of defining, validating and testing executable statecharts [6]. The thirteen participants to the study indicated that the techniques implemented by *Sismic* were easy to use. The received feedback provided evidence that BDD scenarios and runtime monitoring of contracts and property statecharts are beneficial during statechart design. Most participants indicated that they were likely to use these techniques in the future for the purpose of creating new statecharts, or for verifying or modifying existing ones.

Sismic is being used successfully by several companies, notably for model-in-the-loop testing and simulations, for workflow and business process support, and for the execution and validation of concurrent distributed statecharts. We also use *Sismic* in the classroom for teaching executable behavioural modelling to bachelor students.

Since *Sismic* is a pure Python implementation of a statechart execution engine, its performance mainly depends on the performance of the underlying Python engine. By construction, *Sismic* is an interpreter inside an interpreter and as such, has some overhead compared to plain Python code. So far, none of the *Sismic* users has reported any limitation related to performance.

Sismic could be extended with more advanced automated support for statechart testing, such as the generation of statecharts from scenarios, the generation of contracts for a given statechart (similar to the automated generation of contracts over programmes [8]), the generation of tests from contract specifications [9], the use of mutation testing and concolic testing at the level of statecharts [10,11], the detection and improvement of quality problems in statechart and contract specifications, and so on.

5. Conclusions

To conclude, *Sismic* is an open source and extensible Python library with full-fledged support for defining and executing statecharts, and these statecharts can be validated using a portfolio of techniques including unit testing, behaviour-driven development, design by contract, and property statecharts for monitoring of violations of behavioural properties during execution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This paper was carried out in the context of Excellence of Science project O.0157.18F-RG43 “SECO-Assist” financed by Fonds de la Recherche Scientifique—FNRS.

References

[1] Harel D. On visual formalisms. Commun ACM 1988;31(5):514–30. http://dx.doi.org/10.1145/42411.42414.
[2] Beck K. Test-driven development by example. Addison-Wesley; 2002, http://dx.doi.org/10.5555/579193.
[3] North D. Behavior modification: The evolution of behavior-driven development. Better Softw 2006.
[4] Meyer B. Applying "design by contract". IEEE Comput 1992;25(10):40–51. http://dx.doi.org/10.1109/2.161279.
[5] Lu H, Yu S. On the computing power of statecharts. In: Proceedings of the international conference on foundations of computer science (FCS). Citeseer; 2011, p. 1.
[6] Mens T, Decan A, Spanoudakis NI. A method for testing and validating executable statechart models. Softw Syst Model 2019;18(2):837–63. http://dx.doi.org/10.1007/s10270-018-0676-3.
[7] Esmaeilsabzali S, Day NA, Atlee JM, Niu J. Deconstructing the semantics of big-step modelling languages. Requir Eng 2010;15(2):235–65. http://dx.doi.org/10.1007/s00766-010-0102-z.
[8] Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C. The daikon system for dynamic detection of likely invariants. Sci Comput Program 2007;69(1–3):35–45. http://dx.doi.org/10.1016/j.scico.2007.01.015.
[9] Meyer B. Contract-driven development. In: Int'l conf. fundamental approaches to software engineering (FASE). Lect. notes in computer science, vol. 4422, Springer; 2007, p. 11. http://dx.doi.org/10.1007/978-3-540-71289-3_2.
[10] Fabbri SCPF, Maldonado JC, Sugeta T, Masiero PC. Mutation testing applied to validate specifications based on statecharts. In: Int'l symp. software reliability engineering (ISSRE). IEEE Computer Society; 1999, p. 210–9. http://dx.doi.org/10.1109/ISSRE.1999.809326.
[11] Sen K. Concolic testing. In: Int'l conf. automated software engineering. ACM; 2007, p. 571–2. http://dx.doi.org/10.1145/1321631.1321746.