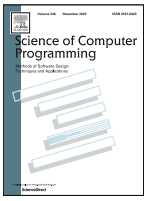
Science of Computer Programming 254 (2026) 103529

Contents lists available at ScienceDirect

Science of Computer Programming

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



Original Software Publication

RebeCaos: A software artefact for Rebeca

José Proença [a,*], Maurice H. ter Beek [b]

[a] CISTER and University of Porto, Rua do Campo Alegre s/n, Porto, 4169-007, Portugal
[b] CNR–ISTI, Via Giuseppe Moruzzi 1, Pisa, 56124, Italy

ARTICLE INFO

Keywords:
Caos
Rebeca
Actors
Time
Web front-end
Reachability analysis

ABSTRACT

We describe RebeCaos: a user-friendly web-based front-end tool for Rebeca, based on the Caos library for Scala. Rebeca is an actor-based language for modelling and analysing concurrent and distributed systems using reactive objects with no shared variables, asynchronous message passing with no blocking when sending and no explicit receiving, and unbounded message buffers for arriving messages. RebeCaos can simulate different operational semantics of (timed) Rebeca, thus facilitating the dissemination and awareness of Rebeca, providing insights into the differences among existing semantics for Rebeca, and supporting quick experimentation of new Rebeca variants (e.g., when the order of received messages is preserved or prioritised). RebeCaos also provides initial reachability analyses for Rebeca models (e.g., the possibility of reaching deadlocks or desirable states).

1. Motivation and significance

The user-friendly web-based front-end tool RebeCaos for the actor-based language Rebeca was first presented during the DisCoTec conference COORDINATION as a tool paper [1], which was awarded with the EAPLS artefact badges Available, Functional and Reusable, and which accompanied a prior Festschrift contribution [2], which we refer to in particular for more details. In this paper, we focus on the software artefact, providing more details on its architecture and on what precisely it offers (cf. Table 1).

This recent artefact is meant to provide an easy entry point to animate the semantics of Rebeca, i.e., ready and easy to use without any installation, providing a small yet extensible core implementation, and rich enough to engage both students and researchers into experimentation and dissemination.

The Reactive objects language (Rebeca) [3–5] is a high-level language designed for modelling and analysing concurrent and distributed systems based on the actor model of computation, which views systems as a collection of autonomous objects (actors) that communicate via asynchronous message passing. Actors or reactive objects (rebecs) constitute its primary modelling components, which are particularly useful for modelling and analysing reactive systems in which components react to incoming messages. Rebecs communicate in a non-blocking fashion via asynchronous message passing between senders and receivers. Each rebec has a set of variables that store values, a set of methods (called message servers), and a message bag (implicitly acting as a buffer between senders and receivers) to store the received messages (along with their arrival times and their deadlines). Operationally, a rebec may take a message (with the least arrival time) from its message bag and execute the corresponding message server. Each rebec operates concurrently and can process one message at a time. Unlike many existing implementations of actors in mainstream languages (e.g., in Erlang/Elixir [6], Scala [7], and Swift [8]), Rebeca is a modelling language, meant to be simpler and more compact, while being expressive enough to reason about the behaviour of actor-based systems.

* Corresponding author.
E-mail addresses: jose.proenca@fc.up.pt (J. Proença), maurice.terbeek@isti.cnr.it (M.H. ter Beek).

https://doi.org/10.1016/j.scico.2026.103529
Received 15 January 2026; Received in revised form 12 May 2026; Accepted 17 June 2026
Available online 2 July 2026

Table 1
Code metadata.

Nr.	Code metadata description	
C1	Current code version	v0.2
C2	Permanent link to code/repository used for this code version	https://github.com/FM-DCC/rebecaos/releases/tag/v0.2
C3	Permanent link to Reproducible Capsule	https://zenodo.org/records/14947780
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Scala & JavaScript
C7	Compilation requirements, operating environments and dependencies	Scala building tools (sbt) & Java Runtime Environment (JRE) $\geq$ 1.7
C8	If available, link to developer documentation/manual	https://doi.org/10.5281/zenodo.14947780 https://github.com/FM-DCC/rebecaos/
C9	Support Email for questions	jose@proenca.org

Throughout the years, several extensions of Rebeca have been introduced for the modelling and analysis of systems from specific domains, among which pRebeca [9] for probabilistic systems, Timed Rebeca [10] for real-time systems, PTRebeca [11] for probabilistic timed systems, and Hybrid Rebeca [12] for cyber-physical systems. In particular, Timed Rebeca extends core Rebeca with a global notion of time [10,13–15], which is achieved by synchronisation of (local) time of the actors (rebecs) involved. In Timed Rebeca, the primitives *delay* and *after* are used to model the progress of time while executing a message server. Timed Rebeca is supported by the tool Afra [16], which offers a comprehensive IDE for specifying and verifying Rebeca models. It unifies the Java artefacts from various Rebeca-related projects and offers tools for model creation, property specification, model checking, and counterexample visualisation. Besides Afra, there is a rich ecosystem of tools around Rebeca [3],[1] including generators for back-end model checkers like SMV [17], mCRL2 [18], and McErlang [19], a dedicated model checker Modere [20], and a more recent Jacco model checker for Java actors [21] based on Rebeca's existing toolset. RebeCaos is our contribution to this toolset. In comparison to the existing tools, RebeCaos is overall easier to use and simpler to deploy (web-based, requiring no installation), easier to extend and more suitable for teaching (small but extensible core implementation, rich enough to engage students into experimentation), but typically worse for analysis and performance (limited analysis features offered, performance not optimised).

The recent Basic Actor-based Language (Babel) [22] is a core actor-based language with a semantics with variation points for communication and coordination that can be instantiated to obtain the concrete semantics of actor-based languages like Rebeca, ABS [23], and Erlang/Akka [24,25]. In particular, the operations of sending and receiving messages are distinguished by variation points based on when a message can be added to the buffer (i.e., based on the buffer's type and capacity) and when it can be consumed from the buffer (e.g., FIFO, EDF (Earliest Deadline First), based on priority or pattern matching, or nondeterministically). RebeCaos allows one to experiment with some of these asynchronous communication policies.

RebeCaos is based on the recently developed the Caos Scala framework (Computer aided design of structural operational semantics) [26]. Caos supports the creation of interactive JavaScript-based websites meant to animate operational semantics. These websites can provide both a quick feedback for developers and good insights to newcomers of a specific language. Caos has been used, e.g., to animate and guide the development of the semantics of choreographic languages [27–30] and reactive systems [31–34], as well as to teach students about the semantics of C-like languages and of a concurrent process calculus.[2]

A survey from 2020 with the participation of 130 formal methods experts—including three Turing Award winners, all five FME Fellowship Award winners, and 17 CAV Award winners—acknowledges the importance of supporting tools for teaching formal methods, since "an overwhelming majority of answers judged the use of tools essential when teaching formal methods" (75.4% of the respondents answered "major role") when asked "whether, and to which extent, students should be exposed to software tools when being taught formal methods" [35, Section 6: Formal Methods in Education]. Moreover, tools "allow students to quickly link theory with practice" [36].

2. Software description

RebeCaos is an online tool that does not require any server. Instead, its functionality is fully compiled into a JavaScript program (from Scala[3] source code) that is loaded by a provided HTML file. This architecture offers a self-contained and quick-to-deploy infrastructure, which can be used with no installation. Section 3 presents illustrative examples, while the impact of this approach is described in Section 4. The rest of this section provides more details about the underlying architecture of RebeCaos (Section 2.1) and about the functionality provided by its widgets (Section 2.2).

2.1. Software architecture

Fig. 1 depicts the overall structure of RebeCaos and its interaction with the Caos libraries. The highlighted nodes are specific to RebeCaos. The bridge between the two is achieved by the CaosConfig.scala file,[4] which extends a Configurator class from Caos

[1] https://rebeca-lang.org/tools
[2] Many examples are listed here: https://github.com/arcalab/caos
[3] https://www.scala-lang.org
[4] https://github.com/FM-DCC/rebecaos/blob/v0.2/src/main/scala/rebecaos/frontend/CaosConfig.scala

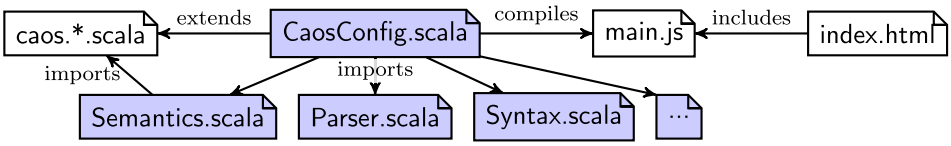


Fig. 1. Architecture of RebeCaos and its interaction with CAOS.

specifying the layout of RebeCaos. More specifically, by specifying (1) which parser is used, (2) a list of examples, (3) a list of widget specifications, and (4) optional documentation for each widget. The parser was built using a library of parser combinators, outputting a data structure representing the syntax of a Rebeca program. The list of examples covers many of the examples from the online repository of Rebeca, as mentioned in Section 3. The analyses are performed by concrete widgets which use *widget constructors* offered by Caos. The main widget constructors used by RebeCaos are listed below.

- **Visualisation** – to calculate a result from a Rebeca program and present it either as plain text or as a Mermaid diagram.[5] We use these to display the number of states and edges (widget "Number of states and edges"), and to present the result from the reachability analysis (widget "Reachability checks").
- **Semantics, step-by-step** – to present how the state of a Rebeca program can evolve, used by the widgets "Run semantics (state's view)" and "Run semantics (sequence chart)". These widgets are parameterised by a Semantics object, which extends a class from Caos that implements a next function calculating the set of next states from a given state.
- **Semantics, all steps** – to use the same Semantics object as above to produce a transition system with all reachable states of the system (bounded by a fixed number of transitions). This transition system can be presented all at once (widget "Build LTS"), or by expanding state by state (widget "Build LTS (explore)").

The current GitHub repository hosting this tool includes a special folder docs with a copy of the index.HTML and supporting files (*.ccs, *.js, etc.), including a snapshot of a compiled JavaScript main.js (cf. Fig. 1). Using the GitHub pages functionality,[6] this folder becomes instantly available by its generated URL from GitHub, namely https://fm-dcc.github.io/rebecaos/. To update the version that users can use, one only needs to update the compiled main.js in the docs folder, supporting quick updates to the online tool.

2.2. Widgets provided by RebeCaos

RebeCaos provides two input widgets and six output widgets, anticipated above. Input widget "Input Rebeca program" allows users to manually write a Rebeca program and "Examples" allows users to load one of the available example programs. The output widgets, reloaded every time they are expanded (by clicking on their headers), are described next.

- "Run semantics (state's view)" and "Run semantics (sequence chart)" interactively ask which reduction should be applied, and depict the current state of the system either using plain text (state's view) or as a sequence chart with the history of interactions (solid arrows) and the pending messages (dashed arrows).
- "Build LTS" applies all possible reductions [2, Section 2.2: Semantics] from the adopted semantics,[7] depicting the state space as a graph, limiting the traversal to a bounded number of transitions.
- "Build LTS (explore)" builds the same LTS as "Build LTS", now interactively, initially presenting only the initial state and its successors—clicking any of these successors adds its own successors.
- "Number of states and edges" counts how many states and edges are found in the traversed LTS (bounded by a maximum number of edges).
- "Reachability checks" traverses the state space searching for a state where a given state properties ϕ holds—these properties can be provided in the input widge by writing, e.g., "`check alice.max == 2;`" to search for states where the local variable `max` of the object `alice` has value 2 (which happens after the `initial` method is executed).

3. Illustrative examples

Fig. 2 shows a screenshot of RebeCaos analysing a small toy example of a bounded ping-pong game, accessible at https://fm-dcc.github.io/rebecaos.

This screenshot includes an input widget in the upper-left corner, and another one with examples immediately below, as well as six output widgets on the right side, only two of them expanded while the others are collapsed.

In this example, two reactive objects of class `Ponger` are created in the main block, called `alice` and `bob`. The `alice` object is initialised with `bob` (the neighbour), `true` (to start the interaction), and `2` (to set lifespan of a chain of interactions). Both objects

[5] A popular JavaScript library for UML-like diagrams: http://mermaid.js.org/

[6] https://docs.github.com/en/pages/

[7] RebeCaos adopted the core semantics of Rebeca with time and dynamic actor creation, but does not include some extensions such as the probabilistic semantics or explicit arrays.

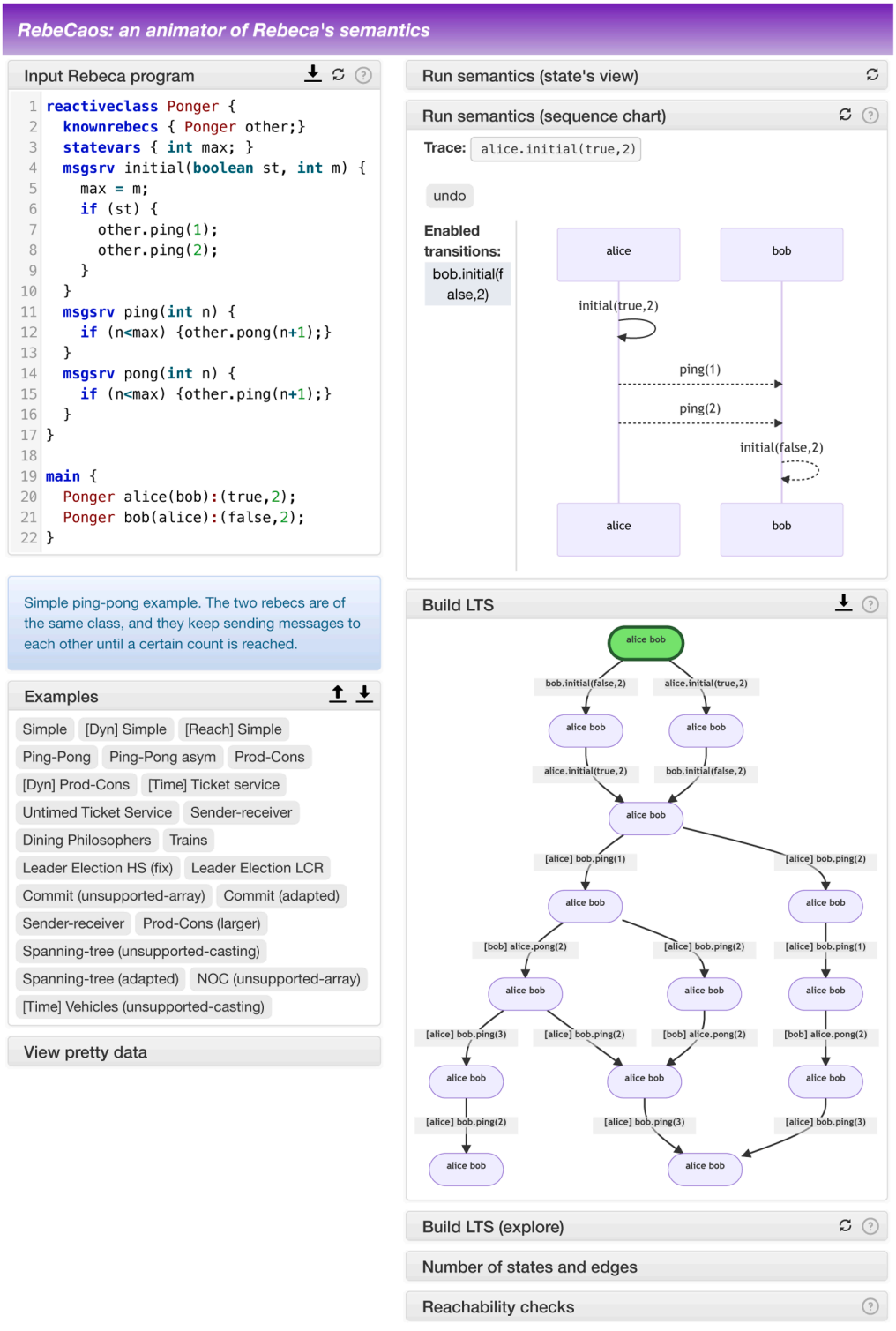


Fig. 2. Screenshot of RebeCaos with a simple ping-pong game.

have a constructor method `initial` that initialises the state and, if set to start, calls the other object twice via the method `ping`. The call is asynchronous, which means that its execution is only started once this object finishes executing the current method body. The sequence diagram on the right depicts the state of the system after one method has been processed, namely the method `initial` by `alice`. At this point, there are three pending messages, marked with dashed lines, namely `ping(1)`, `ping(2)`, and `initial(false,2)`, to be processed by `bob`. Only the method `initial` can now be taken, since it has priority over the remaining methods to initialise the object.

The source code of this and many other examples can be found on the RebeCaos website,[8] among which the following ones.

- The example in Fig. 2, to illustrate how to create and simulate two actors playing the ping-pong game.
- A simpler introductory example, adapted from Hojjat et al. [18] to illustrate the dynamic extension of Rebeca, i.e., support for runtime creation of objects, used in our previous publications introducing RebeCaos [1,2].
- A publisher-consumer example, borrowed from Sirjani [3] and other papers, and its dynamic variation.
- A time-sensitive ticket-service example, borrowed from Khamespanah et al. [15] and other papers, and its untimed version.
- Several other examples included in the online repository of examples,[9] some of which include extensions that are not yet supported by RebeCaos, such as macros and arrays.

4. Impact

We list some advantages of this tool with respect to other Rebeca tools,[10] discussed in the Introduction.

- *No installation needed* – it is sufficient to open a stand-alone website.
- *Quick feedback* – the widgets to animate the semantics are responsive for many useful examples.
- *Compact implementation* – making changes to the syntax and semantics is relatively easy to implement and experiment with.
- *Engaging* – in the sense that it simple to use, e.g., to teach students about Rebeca or to discuss examples or variants with research partners.

This tool is recent, developed in 2025, and it is not yet widely adopted.

Concerning the relatively easy modifications to the syntax and semantics, we make the following remarks. One would need to navigate through the source code and change relevant parts; e.g., small changes to the semantics could be applied to a single function that, given a current state, calculates the next possible reduction steps. More complex changes (e.g., involving new syntactic constructs) may require to update the internal data representation, the parser, the textual representation of the internal data, the evaluation of expressions, the structure of the state of the program, and—again—the semantics. Two examples on how to modify the source code are documented in the accompanying permanent artifact (https://zenodo.org/records/14947780).

5. Conclusions

We presented the RebeCaos software artefact: a user-friendly web-based front-end tool based on the Caos library for Scala. In particular, we described how to use RebeCaos to analyse the behaviour of Rebeca programs, implemented Scala and JavaScript. We expect RebeCaos to be useful for assisting teaching and explaining the insights of Rebeca, as well as for providing a relatively easy and intuitive tool for quick feedback on experiments with new extensions or with variations of the semantics.

In the future, we might extend the syntax and semantics of Rebeca currently supported by RebeCaos with one of the following variants of Rebeca from the literature:

- a means to deal with probabilistic extensions of Rebeca like pRebeca [9] or PTRebeca [11];
- support for the symbolic time representation presented by Khamespanah et al. [15];
- a variation of asynchronous communication policies proposed for Babel by Ghassemi et al. [22].

Alternatively, we could consider entirely new extensions such as an experimental type-checking algorithm for verifying conformity with a given behavioural type.

CRediT authorship contribution statement

José Proença: Writing – review & editing, Writing – original draft, Software, Methodology; **Maurice H. ter Beek:** Writing – review & editing, Conceptualization.

[8] https://www.github.com/fm-dcc/rebecaos
[9] https://rebeca-lang.org/examples
[10] https://rebeca-lang.org/tools

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests.

Given his role as editorial board member for Science of Computer Programming, M.H. ter Beek had no involvement in the peer review of this article and had no access to information regarding its peer review. Full responsibility for the editorial process for this article was delegated to another journal editor. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the CISTER Research Unit (UIDP/UIDB/04234/2020), financed by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology); and by National Funds through the FCT within the project POISE, with reference 2024.15783.PEX (https://doi.org/10.54499/2024.15783.PEX).

References

[1] J. Proença, M.H. ter Beek, RebeCaos, in: C. Di Giusto, A. Ravara (Eds.), Proceedings of the 27th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2025), Vol. 15731 of LNCS, Springer, 2025, pp. 219–229. https://doi.org/10.1007/978-3-031-95589-1_11
[2] M.H. ter Beek, J. Proença, Animating Rebeca, in: E.A. Lee, M.R. Mousavi, C. Talcott (Eds.), Rebeca for Actor Analysis in Action, Vol. 15560 of LNCS, Springer, 2025, pp. 182–194. https://doi.org/10.1007/978-3-031-85134-6_8
[3] M. Sirjani, Rebeca: theory, applications, and tools, in: F.S. de Boer, M.M. Bonsangue, S. Graf, W.P. de Roever (Eds.), Revised Lectures of the 5th International Symposium on Formal Methods for Components and Objects (FMCO 2006), Vol. 4709 of LNCS, Springer, 2006, pp. 102–126. https://doi.org/10.1007/978-3-540-74792-5_5
[4] M. Sirjani, M.M. Jaghoori, Ten years of analyzing actors: Rebeca experience, in: G. Agha, O. Danvy, J. Meseguer (Eds.), Formal Modeling: Actors, Open Systems, Biological Systems, Vol. 7000 of LNCS, Springer, 2011, pp. 20–56. https://doi.org/10.1007/978-3-642-24933-4_3
[5] R. Khosravi, E. Khamespanah, F. Ghassemi, M. Sirjani, Actors upgraded for variability, adaptability, and determinism, in: F.S. de Boer, F. Damiani, R. Hähnle, E.B. Johnsen, E. Kamburjan (Eds.), Active Object Languages: Current Research Trends, Vol. 14360 of LNCS, Springer, 2024, pp. 226–260. https://doi.org/10.1007/978-3-031-51060-1_9
[6] S. Juric, Elixir in Action, Simon and Schuster, 2024.
[7] M. Odersky, L. Spoon, B. Venners, Programming in Scala, 5th ed., Artima Inc, 2021.
[8] M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, I.T. Foster, Swift: a language for distributed parallel scripting, Parallel Comput. 37 (9) (2011) 633–652. https://doi.org/10.1016/J.PARCO.2011.05.005
[9] M. Varshosaz, R. Khosravi, Modeling and verification of probabilistic actor systems using pRebeca, in: T. Aoki, K. Taguchi (Eds.), Proceedings of the 14th International Conference on Formal Engineering Methods (ICFEM 2012), Vol. 7635 of LNCS, Springer, 2012, pp. 135–150. https://doi.org/10.1007/978-3-642-34281-3_12
[10] M. Sirjani, E. Khamespanah, On time actors, in: E. Ábrahám, M.M. Bonsangue, E.B. Johnsen (Eds.), Theory and Practice of Formal Methods, Vol. 9660 of LNCS, Springer, 2016, pp. 373–392. https://doi.org/10.1007/978-3-319-30734-3_25
[11] A. Jafari, E. Khamespanah, M. Sirjani, H. Hermanns, M. Cimini, PTRebeca: modeling and analysis of distributed and asynchronous systems, Sci. Comput. Program. 128 (2016) 22–50. https://doi.org/10.1016/J.SCICO.2016.03.004
[12] I. Jahandideh, F. Ghassemi, M. Sirjani, Hybrid Rebeca: modeling and analyzing of cyber-physical systems, in: R.D. Chamberlain, W. Taha, M. Törngren (Eds.), Revised Selected Papers of the 8th International Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy'18) and the 14th International Workshop on Embedded and Cyber-Physical Systems Education (WESE 2018), Vol. 11615 of LNCS, Springer, 2018, pp. 3–27. https://doi.org/10.1007/978-3-030-23703-5_1
[13] L. Aceto, M. Cimini, A. Ingólfsdóttir, A.H. Reynisson, S.H. Sigurdarson, M. Sirjani, Modelling and simulation of asynchronous real-time systems using Timed Rebeca, in: M.R. Mousavi, A. Ravara (Eds.), Proceedings of the 10th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2011), Vol. 58 of EPTCS, 2011, pp. 1–19. https://doi.org/10.4204/EPTCS.58.1
[14] A.H. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, S.H. Sigurdarson, Modelling and simulation of asynchronous real-time systems using Timed Rebeca, Sci. Comput. Program. 89 (2014) 41–68. https://doi.org/10.1016/J.SCICO.2014.01.008
[15] E. Khamespanah, M. Sirjani, Z. Sabahi-Kaviani, R. Khosravi, M. Izadi, Timed Rebeca schedulability and deadlock freedom analysis using bounded floating time transition system, Sci. Comput. Program. 98 (2015) 184–204. https://doi.org/10.1016/J.SCICO.2014.07.005
[16] E. Khamespanah, M. Sirjani, R. Khosravi, Afra: an Eclipse-based tool with extensible architecture for modeling and model checking of Rebeca family models, in: H. Hojjat, E. Ábrahám (Eds.), Revised Selected Papers of the 10th International Conference on Fundamentals of Software Engineering (FSEN 2023), Vol. 14155 of LNCS, Springer, 2023, pp. 72–87. https://doi.org/10.1007/978-3-031-42441-0_6
[17] M. Sirjani, A. Shali, M.M. Jaghoori, H. Iravanchi, A. Movaghar, A front-end tool for automated abstraction and modular verification of actor-based models, in: Proceedings of the 4th International Conference on Application of Concurrency to System Design (ACSD 2004), IEEE, 2004, pp. 145–150. https://doi.org/10.1109/CSD.2004.1309125
[18] H. Hojjat, M. Sirjani, M.R. Mousavi, J.F. Groote, Sarir: a Rebeca to mCRL2 translator, in: Proceedings of the 7th International Conference on Application of Concurrency to System Design (ACSD 2007), IEEE, 2007, pp. 216–222. https://doi.org/10.1109/ACSD.2007.62
[19] L. Fredlund, H. Svensson, McErlang: a model checker for a distributed functional programming language, in: Proceedings of the 12th International Conference on Functional Programming (ICFP 2007), ACM, 2007, pp. 125–136. https://doi.org/10.1145/1291151.1291171
[20] M.M. Jaghoori, A. Movaghar, M. Sirjani, Modere: the model-checking engine of Rebeca, in: Proceedings of the 21st Symposium on Applied Computing (SAC 2006), ACM, 2006, pp. 1810–1815. https://doi.org/10.1145/1141277.1141704
[21] A. Zakeriyan, E. Khamespanah, M. Sirjani, R. Khosravi, Jacco: more efficient model checking toolset for Java actor programs, in: E.G. Boix, P. Haller, A. Ricci, C.A. Varela (Eds.), Proceedings of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control (AGERE! 2015), ACM, 2015, pp. 37–44. https://doi.org/10.1145/2824815.2824819
[22] F. Ghassemi, M. Sirjani, E. Khamespanah, M. Mirani, H. Hojjat, Transparent actor model, in: Proceedings of the 11th International Conference on Formal Methods in Software Engineering (FormaliSE 2023), IEEE, 2023, pp. 97–107. https://doi.org/10.1109/FORMALISE58978.2023.00018
[23] E.B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, M. Steffen, ABS: a core language for abstract behavioral specification, in: B.K. Aichernig, F.S. de Boer, M.M. Bonsangue (Eds.), Revised Papers of the 9th International Symposium on Formal Methods for Components and Objects (FMCO 2010), Vol. 6957 of LNCS, Springer, 2010, pp. 142–164. https://doi.org/10.1007/978-3-642-25271-6_8
[24] J. Armstrong, R. Virding, M. Williams, Concurrent Programming in Erlang, Prentice Hall, 2nd ed., 1996.
[25] P. Haller, On the integration of the actor model in mainstream technologies: the Scala perspective, in: G.A. Agha, R.H. Bordini, A. Marron, A. Ricci (Eds.), Proceedings of the 2nd Edition on Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions (AGERE! 2012), ACM, 2012, pp. 1–6. https://doi.org/10.1145/2414639.2414641

[26] J. Proença, L. Edixhoven, Caos: a reusable Scala web animator of operational semantics, in: S.-S. Jongmans, A. Lopes (Eds.), Proceedings of the 25th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2023), Vol. 13908 of LNCS, Springer, 2023, pp. 163–171. https://doi.org/10.1007/978-3-031-35361-1_9
[27] L. Edixhoven, S.-S. Jongmans, J. Proença, G. Cledou, Branching pomsets for choreographies, in: C. Aubert, C. Di Giusto, L. Safina, A. Scalas (Eds.), Proceedings of the 15th Interaction and Concurrency Experience (ICE 2022), Vol. 365 of EPTCS, 2022, pp. 37–52. https://doi.org/10.4204/EPTCS.365.3
[28] L. Edixhoven, S.-S. Jongmans, Realisability of branching pomsets, in: S.L. Tapia Tarifa, J. Proença (Eds.), Proceedings of the 18th International Conference on Formal Aspects of Component Software (FACS 2022), Vol. 13712 of LNCS, Springer, 2022, pp. 185–204. https://doi.org/10.1007/978-3-031-20872-0_11
[29] G. Cledou, L. Edixhoven, S.-S. Jongmans, J. Proença, API generation for multiparty session types, revisited and revised using Scala 3, in: K. Ali, J. Vitek (Eds.), Proceedings of the 36th European Conference on Object-Oriented Programming (ECOOP 2022), Vol. 222 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 27:1–27:28. https://doi.org/10.4230/LIPIcs.ECOOP.2022.27
[30] S.-S. Jongmans, J. Proença, ST4MP: a blueprint of multiparty session typing for multilingual programming, in: T. Margaria, B. Steffen (Eds.), Proceedings of the 11th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles (ISoLA 2022), Vol. 13701 of LNCS, Springer, 2022, pp. 460–478. https://doi.org/10.1007/978-3-031-19849-6_26
[31] M.H. ter Beek, G. Cledou, R. Hennicker, J. Proença, Can we communicate? Using dynamic logic to verify team automata, in: M. Chechik, J.-P. Katoen, M. Leucker (Eds.), Proceedings of the 25th International Symposium on Formal Methods (FM 2023), Vol. 14000 of LNCS, Springer, 2023, pp. 122–141. https://doi.org/10.1007/978-3-031-27481-7_9
[32] M.H. ter Beek, R. Hennicker, J. Proença, Team automata: overview and roadmap, in: I. Castellani, F. Tiezzi (Eds.), Proceedings of the 26th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2024), Vol. 14676 of LNCS, Springer, 2024, pp. 161–198. https://doi.org/10.1007/978-3-031-62697-5_10
[33] D. Tinoco, A. Madeira, M.A. Martins, J. Proença, Reactive graphs in action, in: D. Marmsoler, M. Sun (Eds.), Proceedings of the 20th International Conference on Formal Aspects of Component Software (FACS 2024), Vol. 15189 of LNCS, Springer, 2024, pp. 97–105. Cf. Reactive graphs in action (extended version), arXiv:2407.14705[cs.PL], https://doi.org/10.1007/978-3-031-71261-6_6
[34] D. Basile, M.H. ter Beek, J. Proença, Asynchronous team automata, in: A. Sampaio, M. Stoelinga (Eds.), Proceedings of the 27th International Symposium on Formal Methods (FM 2026), Vol. 16557 of LNCS, Springer, 2026, pp. 27–49. https://doi.org/10.1007/978-3-032-26220-2_2
[35] H. Garavel, M.H. ter Beek, J. van de Pol, The 2020 expert survey on formal methods, in: M.H. ter Beek, D. Ničković (Eds.), Proceedings of the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS 2020), Vol. 12327 of LNCS, Springer, 2020, pp. 3–69. https://doi.org/10.1007/978-3-030-58298-2_1
[36] M. ter Beek, M. Broy, B. Dongol, The role of formal methods in computer science education, ACM Inroads 15 (4) (2024) 58–66. https://doi.org/10.1145/3702231