



HAL
open science

State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development

Stefan Klikovits, Rima Al-Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, Dominique Blouin, Etienne Borde, Didier Buchs, Holger Giese, Miguel Goulão, et al.

► **To cite this version:**

Stefan Klikovits, Rima Al-Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, et al.. State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development. [Research Report] COST European Cooperation in Science and Technology. 2019. hal-03168832

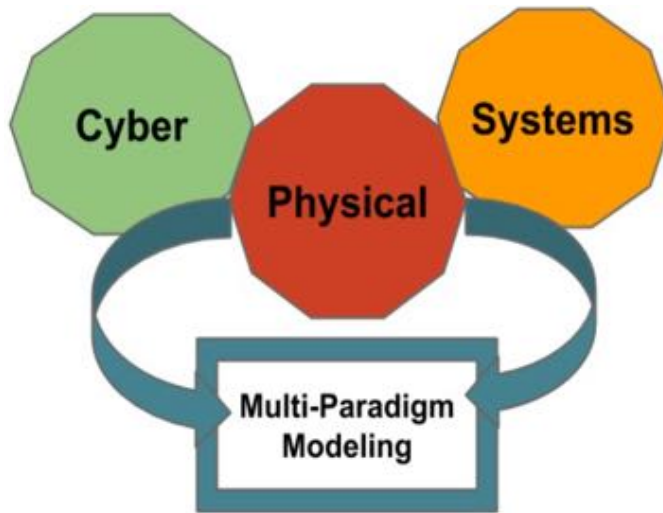
HAL Id: hal-03168832

<https://hal.science/hal-03168832v1>

Submitted on 14 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



ICT COST Action IC1404

State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development

Stefan Klikovits, Rima Al-Ali, Moussa Amrani, Ankica Barišić, Fernando Barros, Dominique Blouin, Etienne Borde, Didier Buchs, Holger Giese, Miguel Goulão, Mauro Iacono, Florin Leon, Eva Navarro, Patrizio Pelliccione, Ken Vanherpen

Deliverable: WG1.1

Core Team

University of Antwerp, Belgium
 New University of Lisbon, Portugal
 Telecom ParisTech, Paris, France
 Hasso-Plattner Inst., Potsdam, Germany
 University of Geneva, Switzerland
 Charles University, Prague, Czech Republic
 University of Manchester, UK
 University of Coimbra, Portugal
 University of Namur, Belgium
 University of Campania Luigi Vanvitelli, Caserta, Italy
 Technical University Asachi of Iasi, Romania
 Chalmers University of Technology, Goteborg, Sweden

Document Info

Deliverable	WG1.1
Dissemination	Restricted
Status	Final
Doc's Lead Partner	Hasso-Plattner Inst.
Date	January 7, 2019
Version	3.0
Pages	86



Contents

1 Introduction	1
2 Structured Catalog of Modelling Languages and Tools	3
2.1 Catalog Structure	4
2.2 Formalisms	5
2.3 Languages	22
2.4 Tools	49
3 Summary and Future Work	70
Bibliography	71
Index	82

1 Introduction

This report presents a substantial catalog of formalisms, modelling languages and tools that are used in the domain of cyber-physical systems modelling. It is part of the deliverable produced by the MPM4CPS's¹ *Working Group 1 (WG1) on Foundations of MPM4CPS*.

Initial WG1 developments prompted the use of ontologies for the analysis of structures and relations of modelling elements. In particular, ontologies are well supported by the OWL formalism and its Protege tool, which allowed to formally capture these modeling elements and their relationships, and to reason about them. Such domain analysis allowed to derive a classification that served as the backbone structure of this catalog.

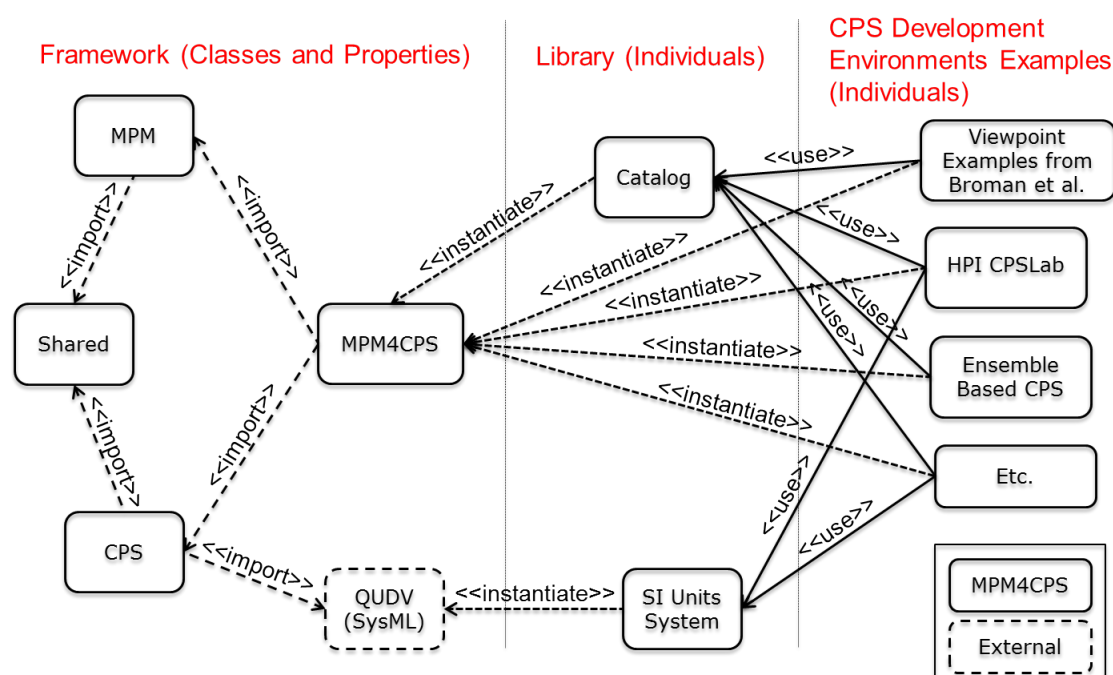


Figure 1.1: Overview of the structure of the MPM4CPS ontology

In particular, figure 1.1 provides an overview of the MPM4CPS ontology (shown in the first column). It emphasises the unification of the two domains of Multi-Paradigm Modelling (MPM) and Cyber-Physical Systems (CPS), which are both foundations for this new emerging domain.

The figure further shows that the catalog (i.e. the contents of this document) is an instantiation of the MPM4CPS domain. The right column of the figure shows few examples of CPS developments that use multi-paradigm modelling. These examples will not be presented in this report, but are elaborated on in the *Framework to Relate / Combine Modeling Languages and Techniques* (see Deliverable D1.2 Giese and Blouin (2016)).

Automatic Catalog Creation: As stated, the individual formalisms, languages and tools of this catalog are based upon the MPM4CPS ontology. As one can easily imagine, many of the over 170 individual representatives of these concepts contain numerous connections to other individuals. In general, a tight network of dependencies, compatibilities and other relations was discovered. To fully exploit modern technologies, the catalog entries were entered directly into the ontology, so that the connections could be defined. This catalog is generated directly from these ontology sources and thus shows one view of this ontology. As an interactive, searchable

¹ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)



and visual depiction of this data provides a more comprehensive view than this static generation, we are in the process of publishing the digital ontology alongside the final report.

Usage of this catalog This catalog serves as a valuable source of information and can be used as a reference to look up data about a large number of current modelling technologies and approaches. The authors tried to add further references to each entry, such that formalisms show scientific references or sources of introduction material, and tool implementations are annotated with a hyperlink to the tool's webpage, where it can be downloaded from.

The catalog is structured as follows: Chapter 2 presents the list of identified formalisms (Section 2.2), modelling languages (Section 2.3) and modelling tools (Section 2.4). The list items each have a short description, extended by references and links to further resources to guide interested readers towards further information.



2 Structured Catalog of Modelling Languages and Tools

The MPM4CPS domain merges concepts from multi-paradigm modelling (MPM) and applies them onto the development of cyber-physical systems. As an extension of the well-known field of embedded systems development, the CPS discipline emerges from a widely studied field and thus builds upon its previous insights, development and modelling approaches and languages and tools.

On the other hand, CPS is a very recent field that embraces new ideas and modern paradigms. These additions created diverging branches from existing standards and new target audiences and domain actors adopted multiple approaches: the result is that different stakeholders use different definitions for their approaches, with an evident dispersion of effort and a significant introduction of unnecessary conceptual entropy in the field in both literature and practice, that may be dominated by a substanding unifying approach suitable to keep beneficial diversity. Consequently, MPM's aim is to further the diversity of topics by taking a very elastic approach and supporting generic (informal) definitions and merging the “best formalisms, languages and tools” for each point of view. This stance however resulted in the creation of tools that were generated based on different needs and from different points of view, spanning from small-scale, theoretical and hardware-centric approaches to robust, practical, industrial and large-scale applications, and numerous other solutions inbetween.

This chaotic richness of tools is further founded by the heterogeneity of users and backgrounds that are involved in the MPM4CPS processes. MPM4CPS processes mirror the intrinsic complexity of their systems and diversity of their users. These come from and work in different domains, each bringing their specialised point of view, heritage and methods of their respective domain. Such methods belong to different phases of the design, development and assessment cycle, and stem from various research fields, with particular perspectives on the same problems and different views on similar systems. As a result, specifications of tools are often divergent, even if not contrasting, as they are driven by different purposes, towards different goals, and on different methodological foundations.

This catalog aims to serve as a small collection of the most commonly employed tools, modelling languages and formalisms in the domain. The catalog's initial purpose prescribed to capture current and actively used means to model CPS. However, it quickly became evident that the domain's vastness cannot be captured entirely in this document. Too many tools and languages have been developed and trialed in many experiments and evaluated on numerous case studies. Instead, the decision was made to initially create a description of the most commonly known and used representatives of each category, and, in a second iteration, add representative usage reports and evaluations.

The catalog is by no means attempting to be complete or even exhaustive, as the described vastness and the speed of development, adaptation and change of the discipline renders such a goal unreachable. What it does however is to serve as a reference for newcomers who wish to learn more about a particular formalism, language or tool. The catalog further provides references to additional information resources such as experience reports, case studies and analyses.

Catalog entries has been selected by experts in their respective fields and participants of the MPM4CPS's WG1 on Foundations. Collectively, they serve as a current snapshot of the main and most important concepts, but – similar to other lexica and reference sources – has to be extended, revisited and updated in regular intervals (as intended by the members of WG1).

2.1 Catalog Structure

The structure of the catalog is inspired by the framework proposed by Broman et al. (2012), which distinguishes between viewpoints, formalisms, and languages and tools. It defines that stakeholders have viewpoints which are expressed using formalisms. As formalisms are mathematical structures (i.e. without a concrete syntax), they require languages and tools that implement them. Figure 2.1 depicts this framework visually. Evidently, modern tools often support several languages, and each language can merge several formalisms. For example, Papyrus is a tool that supports the creation of different Unified Modeling Language (UML) diagram types such as *class diagrams*, *sequence diagrams*, etc. Each diagram is based upon a different formalism.

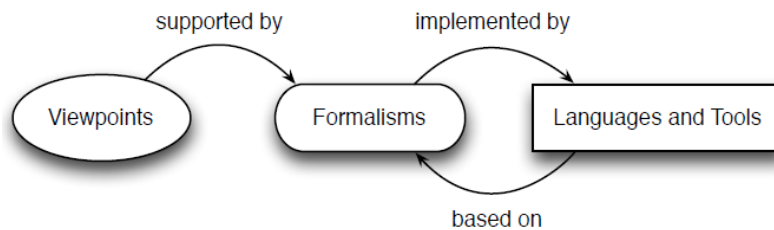


Figure 2.1: Framework for Viewpoints, Formalisms, Languages and Tools (from Broman et al. (2012))

The catalog lists its entries in a pre-defined order, starting from the most theoretical (formalisms) to the most practical (tools). In some situations, the classification of individual entries is not easily possible. For example, *Petri nets* are a family of formalisms that are well-suited for the expression of concurrency. There are various “flavours” of Petri nets that allow the expression of different systems. As they are closely related though, the catalog authors choose to list them as individual formalisms, even though one might argue that *Colored Petri nets* are a formalism by themselves. Another issue arises with the classification of Petri nets, as, when they were introduced, Petri nets were equipped with a visual notation syntax, which nowadays is used interchangeably with the mathematical formalism. This catalog thus lists two different entries for “Petri nets” (i.e. the formalism) and “Petri nets diagram” (i.e. its graphical syntax). The reason behind this is that there exists for example the *PNML* (Petri Net Markup Language) format, which is a common exchange format for Petri net diagrams that can be easily read and written by tools. It is thus another language (with a concrete, textual syntax) for the Petri nets formalism. Finally, the tools section (Section 2.4) provides a list of tools that can be used to create, edit, and interact with models. For example, the section contains an entry for *StrataGEM*, a common tool for the model-checking of concurrent models such as Petri nets.



2.2 Formalisms

The following subsections present the most commonly used formalisms for cyber-physical systems development.

2.2.1 Automata-based Formalisms

2.2.1.1 Abstract State Machines

Abstract state machines are a formalism that model the states and their transitions. ASM are an extension of Finite State Machines (FSM) and allow the representation of states as arbitrary data types. The formalism aims to bridge the gap between human-readable, easily understandable specifications and machine-executable code. Application areas of ASM are for example the modelling and verification of programming languages, protocols, embedded systems, etc.

Implementing Languages

Abstract State Machines is a formalism for the following languages:

- AADL (see section 2.3.3)
- AsmL (see section 2.3.9)
- AsmetaL (see section 2.3.8)
- Gofer (see section 2.3.21)

References

- Börger, E. and W. Schulte (1999), *A Programmer Friendly Modular Definition of the Semantics of Java*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 353–404, ISBN 978-3-540-48737-1, doi: 10.1007/3-540-48737-9_10.
https://doi.org/10.1007/3-540-48737-9_10
- Borger, E. (2005), Abstract state machines and high-level system design and analysis, *Theoretical Computer Science*, **336**, 2, pp. 205 – 207, ISSN 0304-3975
- Börger, E. and R. Stärk (2003), Springer Berlin Heidelberg, Berlin, Heidelberg.
<https://doi.org/10.1007/978-3-642-18216-7>

2.2.1.2 Cellular Automata

Cellular Automata (CA) is a popular formalism for the idealisation of physical systems. A system consists of an n-dimensional lattice of uniform “cells”. Each cell is an automaton, whose behaviour (discrete state transitions) is defined by its own state, and the states of cells in its “neighbourhood” (the cells with a certain distance, e.g. direct neighbours). The transitions of all CA in the system are executed at the same time. By representing automaton behaviour using relatively simple rules, highly complex system behaviour can be expressed. Since their introduction in the 1950ies, cellular automata had various waves of popularity. Recently they are used to model and study emergent behaviour in very large and highly complex systems consisting of uniform units. One of the most popular CA case studies is Conway’s “Game of Life”.

Implementing Languages

None

References

- Chopard, B. (2012), *Cellular Automata Modeling of Physical Systems*, Springer New York, New York, NY, pp. 407–433, ISBN 978-1-4614-1800-9, doi: 10.1007/978-1-4614-1800-9_27.
https://doi.org/10.1007/978-1-4614-1800-9_27



- (2018), Cellular-Automata Website, <https://plato.stanford.edu/entries/cellular-automata/>
- Athanassopoulos, S., C. Kaklamanis, G. Kalfountzos and E. Papaioannou (2012), Cellular Automata for Topology Control in Wireless Sensor Networks Using Matlab, in *Proceedings of the 44th IEEE Conference on Decision and Control*, volume 164, volume 164

2.2.1.3 Hybrid Automata-based Formalisms

2.2.1.4 Hybrid Automata

A finite state automaton is a computational abstraction of the transitions of a system between discrete states or locations (on and off, for instance). A hybrid automaton, additionally, considers dynamical evolution over time in each location. This dynamical evolution is represented by a dynamical system. Depending on the nature of the dynamics of this system, different types of hybrid automata are defined; the main ones are explained as follows. Hybrid automata is one of the many existing modelling frameworks of hybrid systems.

Implementing Languages

Hybrid Automata is a formalism for the following languages:

- Stateflow Language (see section 2.3.64)
- Scilab Language (see section 2.3.54)
- PyDSTool Language (see section 2.3.50)
- NuSMV Language (see section 2.3.36)
- Ptolemy Language (see section 2.3.49)
- Modelica (see section 2.3.32)

References

- Lygeros, J. and M. Prandini (2010), Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications, *European Journal of Control*, **16**, 6, pp. 583 – 594, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.583-594>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706889>
- Abate, A., J.-P. Katoen, J. Lygeros and M. Prandini (2010), Approximate Model Checking of Stochastic Hybrid Systems, *European Journal of Control*, **16**, 6, pp. 624 – 641, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.624-641>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706919>

2.2.1.5 I/O Hybrid Automata (Hybrid Input/Output Automata)

I/O hybrid automata are a class of hybrid automata that include asynchronous distributed system components. The automaton's transitions are annotated with input actions and output actions, which are executed at when triggering a transition.

Implementing Languages

None

References

- Lynch, N. A. and M. R. Tuttle (1989), An introduction to input/output automata, *CWI Quarterly*, **2**, pp. 219–246
- Lynch, N., R. Segala and F. Vaandrager (2003), Hybrid I/O automata, *Information and Computation*, **185**, 1, pp. 105 – 157, ISSN 0890-5401, doi: <https://doi.org/10.1016/S0890->



5401(03)00067-1.

<http://www.sciencedirect.com/science/article/pii/S0890540103000671>

2.2.1.6 Linear Hybrid Automata

A linear hybrid automaton is a hybrid automaton where the dynamical system within each location or discrete state is linear.

Implementing Languages

Linear Hybrid Automata is a formalism for the following languages:

- Stateflow Language (see section 2.3.64)
- Scilab Language (see section 2.3.54)
- PyDSTool Language (see section 2.3.50)
- NuSMV Language (see section 2.3.36)
- Ptolemy Language (see section 2.3.49)
- Modelica (see section 2.3.32)

References

- Lygeros, J. and M. Prandini (2010), Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications, *European Journal of Control*, **16**, 6, pp. 583 – 594, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.583-594>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706889>
- Abate, A., J.-P. Katoen, J. Lygeros and M. Prandini (2010), Approximate Model Checking of Stochastic Hybrid Systems, *European Journal of Control*, **16**, 6, pp. 624 – 641, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.624-641>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706919>

2.2.1.7 Non-Linear Hybrid Automata

A nonlinear hybrid automaton is a hybrid automaton where the dynamical system within each location or discrete state is nonlinear. The studies on nonlinear hybrid automata are much less than in linear hybrid automata due to the complexity of the dynamical behaviours involved.

Implementing Languages

Non-Linear Hybrid Automata is a formalism for the following languages:

- Stateflow Language (see section 2.3.64)
- Scilab Language (see section 2.3.54)
- PyDSTool Language (see section 2.3.50)
- NuSMV Language (see section 2.3.36)
- Ptolemy Language (see section 2.3.49)
- Modelica (see section 2.3.32)

References

- Lygeros, J. and M. Prandini (2010), Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications, *European Journal of Control*, **16**, 6, pp. 583 – 594, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.583-594>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706889>



- Abate, A., J.-P. Katoen, J. Lygeros and M. Prandini (2010), Approximate Model Checking of Stochastic Hybrid Systems, *European Journal of Control*, **16**, 6, pp. 624 – 641, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.624-641>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706919>

2.2.1.8 PTAs (Priced/Probabilistic Timed Automata)

A priced timed automaton is a timed automaton where transitions and locations are annotated with real-valued costs and cost rates, respectively. The global cost increases according to the cost rate of the current location and “jumps” when taking a transition (according to the cost annotation of the transition).

Note, that the cost rate of each location can be different, but cannot be negative.

Implementing Languages

PTAs is a formalism for the following languages:

- PRISM Language (see section 2.3.44)

References

- Behrmann, G., K. G. Larsen and J. I. Rasmussen (2005), Priced Timed Automata: Algorithms and Applications, in *Formal Methods for Components and Objects*, Eds. F. S. de Boer, M. M. Bonsangue, S. Graf and W.-P. de Roever, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 162–182, ISBN 978-3-540-31939-9

2.2.1.9 Stochastic Hybrid Automata

A stochastic hybrid automaton is a class of non-deterministic hybrid automaton where uncertainty is introduced in the system evolution and the control inputs. The main differences with respect to a hybrid automaton are: 1) the initial state (initial discrete location and initial continuous state vector) is chosen at random, 2) the continuous state evolves following stochastic differential equations, 3) there are two types of discrete transitions between locations that give rise to two types of guards conditions: forced transitions and spontaneous transitions; spontaneous transitions are triggered by stochastic events.

Implementing Languages

Stochastic Hybrid Automata is a formalism for the following languages:

- Stateflow Language (see section 2.3.64)
- Scilab Language (see section 2.3.54)
- PyDSTool Language (see section 2.3.50)
- Simulink Language (see section 2.3.59)
- Ptolemy Language (see section 2.3.49)
- Modelica (see section 2.3.32)

References

- Lygeros, J. and M. Prandini (2010), Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications, *European Journal of Control*, **16**, 6, pp. 583 – 594, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.583-594>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706889>
- Abate, A., J.-P. Katoen, J. Lygeros and M. Prandini (2010), Approximate Model Checking of Stochastic Hybrid Systems, *European Journal of Control*, **16**, 6, pp. 624 – 641, ISSN



0947-3580, doi: <https://doi.org/10.3166/ejc.16.624-641>.

<http://www.sciencedirect.com/science/article/pii/S0947358010706919>

2.2.1.10 Stochastic Timed Automata

Stochastic timed automata extend the TA formalism by adding probabilistic concepts, which allow the stochastic selection of enabled transitions and delays (i.e. time spent within a location).

Implementing Languages

Stochastic Timed Automata is a formalism for the following languages:

- UPPAAL SMC Language (see section 2.3.72)

References

- Bertrand, N., P. Bouyer, T. Brihaye, Q. Menet, C. Baier, M. Groesser and M. Jurdzinski (2014), Stochastic timed automata, *Logical Methods in Computer Science*, **10**, 4, p. 73, doi: 10.2168/LMCS-10(4:6)2014.
<https://hal.inria.fr/hal-01102368>

2.2.1.11 Timed Automata

Timed automata (TA) is a formalism that merges the concepts of discrete automata theory with the continuous evolution of variables (“clocks”). Each clock progressively increases its value at a constant rate (i.e. 1). Transition guards can be defined by comparing the clocks’ values to constants. Further, each clock can be reset to zero when firing transitions.

Timed automata can be seen as a specialisation of hybrid automata, where all clock rates are constant and equal, transition guards only allow linear comparisons of clocks and transitions (hybrid automata jumps) can only reset clock values to zero.

Many extensions of timed automata have been proposed, such as stopwatch automata (the rate can be temporarily set to 0), and hourglass automata (the clocks’ rate can be reversed to -1).

Implementing Languages

Timed Automata is a formalism for the following languages:

- HyTech Language (see section 2.3.22)
- Kronos Language (see section 2.3.26)
- UPPAAL Language (see section 2.3.71)

References

- Alur, R. and D. L. Dill (1994), A Theory of Timed Automata, *Theor. Comput. Sci.*, **126**, 2, pp. 183–235, ISSN 0304-3975, doi: 10.1016/0304-3975(94)90010-8.
[http://dx.doi.org/10.1016/0304-3975\(94\)90010-8](http://dx.doi.org/10.1016/0304-3975(94)90010-8)
- Cassez, F. and K. Larsen (2000), The Impressive Power of Stopwatches, in *International Conference on Concurrency Theory*, Springer, pp. 138–152
- Bengtsson, J. and W. Yi (2004), *Timed Automata: Semantics, Algorithms and Tools*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 87–124, ISBN 978-3-540-27755-2, doi: 10.1007/978-3-540-27755-2_3.
https://doi.org/10.1007/978-3-540-27755-2_3
- Osada, Y., T. French, M. Reynolds and H. Smallbone (2014), Hourglass Automata, *Electronic Proceedings in Theoretical Computer Science*, **161**, pp. 175–188, ISSN 2075-2180, doi: 10.4204/EPTCS.161.16



2.2.1.12 Timed I/O Automata (Timed Input/Output Automata)

The Timed I/O Automata formalism is an extension of interface automata to include timing information. The automaton's transitions are annotated with input actions and output actions, which are executed at when triggering a transition.

Implementing Languages

None

References

- Kaynar, D. K., N. Lynch, R. Segala and F. Vaandrager (2010), The theory of timed I/O automata, *Synthesis Lectures on Distributed Computing Theory*, **1**, 1, pp. 1–137
- David, A., K. G. Larsen, A. Legay, U. Nyman and A. Wasowski (2010), Timed I/O Automata: A Complete Specification Theory for Real-time Systems, in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ACM, New York, NY, USA, HSCC '10, pp. 91–100, ISBN 978-1-60558-955-8, doi: 10.1145/1755952.1755967. <http://doi.acm.org/10.1145/1755952.1755967>

2.2.2 Flow-based Formalisms

2.2.2.1 Data Flow

DataFlow describes an computation model that resembles a graph. Each node represents a function (usually called actor) and each arc represents a communication channel between two functions. All the inputs of a function must be present to start its execution. In synchronous dataflow modelling the “synchronous hypothesis” states that the computation of each function and the communication between two functions is infinitely fast (or instantaneous).

Given this hypothesis, SDF provides a sound model of computation with predictable performance, properties verification methods (liveness, deadlock freedom), and predictable buffering.

The practical approach relies on the fact that the computation of a node starts only when the execution of all its predecessors is finished. This requires that the graph topology is loop-free. Since SDF are usually executed in periodic tasks, the synchronous hypothesis is usually considered to be verified as the worst case response time of the graph is smaller than the task's period.

This type of model is well-suited to digital signal processing and communications systems which often process an endless supply of data. It has been successfully applied in the domain of safety critical embedded systems. It has also been characterised or extended into homogeneous data flow graphs, cyclo-static data flow graphs, scenario-aware data flow graphs, affine data flow graphs.

Implementing Languages

Data Flow is a formalism for the following languages:

- AADL (see section 2.3.3)
- Zélus (see section 2.3.76)
- Lustre (see section 2.3.28)
- Signal (see section 2.3.55)

References



- Murthy, P. K. and E. A. Lee (2002), Multidimensional synchronous dataflow, *IEEE Transactions on Signal Processing*, **50**, 8, pp. 2064–2079, ISSN 1053-587X, doi: 10.1109/TSP.2002.800830

2.2.2.2 HyFlow (Hybrid Flow System Specification)

HyFlow provides a formal description of dynamic topology hybrid systems Barros (2003). This formalism can model systems that exhibit both continuous and discrete behaviors while relying on a digital computer representation. HyFlow supports multisampling as a first order operator, enabling both time and component varying sampling, making it suitable for representing sampled-based systems, like digital controllers and filters. HyFlow provides also an extrapolation operator that enables an error free representation of continuous signals. Multisampling can be used for achieving an explicit representation of asynchronous adaptive step-size differential equations integrators Barros (2018). HyFlow provides an integrative framework for combining models expressed in different modeling paradigms. In particular, fluid stochastic Petri-nets Barros (2015) and geometric integrators Barros (2018), for example, can be represented in the HyFlow formalism, enabling its seamless integration with other HyFlow models. HyFlow sampling provides an expressive operator for making the connection of computer-based systems with real-time systems, since sampling is, in many cases, the most convenient operator to interact with continuous signals. HyFlow supports modular and hierarchical models providing deterministic semantics for model composition and co-simulation Barros (2008).

Implementing Languages

None

References

- Barros, F. J. (2018), Modular representation of asynchronous geometric integrators with support for dynamic topology, *SIMULATION*, **94**, 3, pp. 259–274
- Barros, F. J. (2008), Semantics of Dynamic Structure Event-based Systems, in *Proceedings of the Second International Conference on Distributed Event-based Systems*, ACM, New York, NY, USA, DEBS '08, pp. 245–252, ISBN 978-1-60558-090-6, doi: 10.1145/1385989.1386020.
<http://doi.acm.org/10.1145/1385989.1386020>
- Barros, F. J. (2003), Dynamic Structure Multiparadigm Modeling and Simulation, *ACM Trans. Model. Comput. Simul.*, **13**, 3, pp. 259–275, ISSN 1049-3301, doi: 10.1145/937332.937335.
<http://doi.acm.org/10.1145/937332.937335>
- Barros, F. (2015), A Modular Representation of Fluid Stochastic Petri Nets, in *Symposium on Theory of Modeling and Simulation*

2.2.3 Logic-based Formalisms

2.2.3.1 CTL (Computation Tree Logic)

CTL is a branching-time temporal logic that allows the analysis of tree-like structures. It allows the analysis of various future evolution paths to answer questions whether there is a possibility that an event might or will happen. Various model checkers exist that allow the verification of the reachability and liveness of certain events (or event sequences).

Implementing Languages

CTL is a formalism for the following languages:

- NuSMV Language (see section 2.3.36)



References

- Huth, M. and M. Ryan (2004), *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, 2 edition, doi: 10.1017/CBO9780511810275

2.2.3.2 First Order Logic

First-Order Logic (FOL, also “First-Order Predicate Calculus”) is a logical formalism constituted of terms and formulas. A term is either a variable, a function symbol or a predicate symbol. A formula is constituted of formulas built over terms combined with the usual Boolean operators (negation, conjunction and so on) and both existential and universal quantifiers. First-Order Logic formulas are interpreted on a (finite) domain, and possess a sound and complete calculus, making it automatable for reasoning Kleene (2002).

Implementing Languages

First Order Logic is a formalism for the following languages:

- SMT-LIB (see section 2.3.61)

References

- Kleene, S. C. (2002), *Mathematical Logic*, Wiley

2.2.3.3 Linear Temporal Logic

LTL is a representative of the family of temporal logics that analyses the future evolution of paths. It allows the testing of one individual branch (as opposed to CTL).

Implementing Languages

Linear Temporal Logic is a formalism for the following languages:

- Kronos Language (see section 2.3.26)
- PSC (see section 2.3.47)

References

- Huth, M. and M. Dermot Ryan (2000), *Logic in computer science - modelling and reasoning about systems.*, ISBN 978-0-521-65602-3

2.2.3.4 Temporal Logic

A temporal logic (also tense logic) system is a formal mathematical system that allows the reasoning about terms and their truth value related to time. It is an extension of first-order logic and allows the expression of statements such as "I always model CPS", "I will eventually model a CPS" and "I will not model CPS until I find a good formalism".

Temporal logic is frequently used in formal verification, where it is used to formally express requirements (e.g. the reaching of a certain state). Popular temporal logic systems are for example the linear temporal logic (LTL), computation tree logic (CTL) and Hennessy-Milner logic (HML).

Implementing Languages

Temporal Logic is a formalism for the following languages:

- NuSMV Language (see section 2.3.36)

References

- Pnueli, A. (1977), The Temporal Logic of Programs, in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Washington, DC,



USA, SFCS '77, pp. 46–57, doi: 10.1109/SFCS.1977.32.
<https://doi.org/10.1109/SFCS.1977.32>

2.2.4 Petri Net-based Formalisms

2.2.4.1 High Level Petri Nets

This large class rely on another modeling framework devoted to the description of the data attached to the tokens, and the expression that must be assigned to arcs describing the computation that must be done for satisfying the pre and post conditions. Among several dialect of these class we can cite algebraic Petri Nets and coloured Petri Nets.

Implementing Languages

High Level Petri Nets is a formalism for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)

References

2.2.4.2 Petri Net (Place Transition Nets)

A Petri net (also known as a place/transition net or P/T net) is a formalism for the description of dynamic discrete systems. It is particularly suited to the modeling of systems where distribution, concurrency and non-determinism is important to describe and analyze. The mathematical foundation of Petri nets has led to several development to make possible complex analysis of systems described by Petri nets. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, represented by bars or black rectangles) and places (i.e. conditions or resources, represented by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Petri nets were invented, according to some sources, in August 1939 by Carl Adam Petri at the age of 13 for the purpose of describing chemical processes.

Petri nets offer a graphical notation which is of particular interest to the engineer analyzing systems, natural pattern of design are existing in Petri nets such as causality, choice, independence and resource management.

There are attempts for several formalisms Like UML activity diagrams, Business Process Model and Notation and EPCs to have semantics in term of translation into Petri nets. These translations have been used to provide a precise semantics to such formalisms as well as to develop tools for their simulation and analysis.

It has been observed that in practice Petri Nets lack of modeling power for some modeling dimensions such as: timing aspects, data structures, stochastic processes and event priority. In general the formal analysis tool need different techniques if new modeling dimensions are introduced. So tools are generally different and adapted to the specific extensions.

Petri Nets have several extensions which are concretised in the following formalisms, it must be noted that the computing power of such extensions can change according to the new concepts introduced in the notations (i.e. time) and then the analysability of the models is sometime much more difficult or limited to extension with finiteness assumptions (i.e. high level nets):

Implementing Languages

Petri Net is a formalism for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)



- FIACRE (see section 2.3.18)

References

- CPN preserve useful properties of Petri nets and at the same time extend initial formalism to allow the distinction between tokens. Coloured Petri Nets allow tokens to have a data value attached to them. This attached data value is called token color.

https://en.wikipedia.org/wiki/Coloured_Petri_net

2.2.4.3 Petri Nets with Priority

This simple extension bring new semantics to the transition firing in order to solve conflict. Depending on the variant it can lead to more powerful models which are Turing complete.

Implementing Languages

Petri Nets with Priority is a formalism for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)

References

2.2.4.4 Stochastic Petri Nets

The firing of transitions follows a probabilistic distribution on time duration or frequency of transitions firing when there is a conflict.

Implementing Languages

Stochastic Petri Nets is a formalism for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)

References

2.2.4.5 Timed Petri Nets

Timed Petri Nets introduce time as a modeling dimension. This is reflected in aspect such as transition duration or possible interval for firing transitions. Generally these classes bring new problems for analysing them, in particular the time-line which has no limit in the future and the density of the time.

Implementing Languages

Timed Petri Nets is a formalism for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)
- FIACRE (see section 2.3.18)

References

- Abdulla, P. A. and A. Nylén (2001), Timed Petri Nets and BQOs, in *Applications and Theory of Petri Nets 2001*, Eds. J.-M. Colom and M. Koutny, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 53–70, ISBN 978-3-540-45740-4
- Berthomieu, B. and M. Diaz (1991), Modeling and verification of time dependent systems using time Petri nets, *IEEE Transactions on Software Engineering*, **17**, 3, pp. 259–273, ISSN 0098-5589, doi: 10.1109/32.75415



2.2.5 Other Formalisms

2.2.5.1 Bayesian Networks

Bayesian networks are a formalism that establishes a probabilistic graph model. The vertices of the directed acyclic graph represent random variable probabilities and edges between them represent probabilistic dependencies. Hence each vertex depends on its predecessors. Bayesian networks are well-suited to create simple computations for models with many variables. They can further be used to encode hypotheses, hypotheses, beliefs, and latent variables.

Implementing Languages

None

References

- Niedermayer, D. (2008), *An Introduction to Bayesian Networks and Their Contemporary Applications*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 117–130, ISBN 978-3-540-85066-3, doi: 10.1007/978-3-540-85066-3_5.
https://doi.org/10.1007/978-3-540-85066-3_5

2.2.5.2 Bond Graphs

Bond graphs focus on the modelling of energy flows. The formalism recognises that different energy domains (electrical, hydraulic, mechanical) can be similarly represented and that the dual forces of "flow" (e.g. current, velocity) and "effort" (e.g. voltage, force) in combination create power. Bond graphs models represent idealised energy flows. Usually the same notation is used for all energy domains and domains switches are modelled using e.g. gyrators and transformers. The formalism is actively used in engineering domains where it serves good purpose of prototyping the energy flow within a system.

Implementing Languages

Bond Graphs is a formalism for the following languages:

- 20-sim Bond Graph (see section 2.3.2)

References

- Paynter, H. M. (1961), *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, MA
- Broenink, J. F. (1999), Introduction to Physical Systems Modelling with Bond Graphs, in *in the SiE whitebook on Simulation Methodologies*, p. 31

2.2.5.3 Causal Block Diagrams

Causal block diagrams are a visual modelling formalism that connects operation blocks in a graph structure. The graph's operation block nodes represent computations or transformations of signals, edges model the signals themselves. Typical operation blocks are algebraic functions (e.g. sum, product) but can also represent timed functionality (e.g. delays, integration or derivatives). The formalism is used in many visual modelling tools (e.g. Simulink) and graphical programming languages (e.g. LabVIEW).

Implementing Languages

Causal Block Diagrams is a formalism for the following languages:

- LabVIEW Language (see section 2.3.27)
- Simulink Language (see section 2.3.59)

References



- Gomes, C., J. Denil and H. Vangheluwe (2016), Causal-Block Diagrams, Technical report. <http://msdl.cs.mcgill.ca/people/claudio/pub/Gomes2016a.pdf>
- Denckla, B. and P. J. Mosterman (2005), Formalizing Causal Block Diagrams for Modeling a Class of Hybrid Dynamic Systems, in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4193–4198, ISSN 0191-2216, doi: 10.1109/CDC.2005.1582820

2.2.5.4 Complex Networks

A complex network is a large number of interdependent systems connected in a nontrivial and non-regular manner. The interconnection of these systems produces emergent properties or behaviours which are not present in the isolated systems: this is called self-organisation, collective behaviour. There are different representations for complex networks. These models are typically based on graph theory (nodes connected with links). The main models for complex networks are: random-graph networks, small-world networks, scale-free networks. Each of these models have different topological (structural) features, analysed with tools from statistical physics.

Implementing Languages

Complex Networks is a formalism for the following languages:

- Visone Language (see section 2.3.74)
- Small-World Networks (see section 2.3.60)
- NetworkX Language (see section 2.3.35)
- Pajek Language (see section 2.3.40)
- Random-Graph Networks (see section 2.3.51)
- iGraph Language (see section 2.3.23)
- Mathematica Language (see section 2.3.30)
- Cytoscape Language (see section 2.3.13)
- Network Workbench Language (see section 2.3.34)

References

- Navarro-Lopez, E. (2013), *DYVERSE: From formal verification to biologically-inspired real-time self-organizing systems*, pp. pp. 301–346, ISBN ISBN-10:1439883270
- H. Strogatz, S. (2001), Exploring Complex Networks, *Nature*, **410**, doi: 10.1038/410268a0. <http://www.nature.com/nature/journal/v410/n6825/pdf/410268a0.pdf>
- Barrat, A., M. Barthelemy and A. Vespignani (2008), *Dynamical Processes on Complex Networks*, Cambridge University Press, doi: 10.1017/CBO9780511791383
- Barabási, A.-L. and R. Albert (1999), Emergence of Scaling in Random Networks, *Science*, **286**, 5439, pp. 509–512, ISSN 0036-8075, doi: 10.1126/science.286.5439.509. <http://science.sciencemag.org/content/286/5439/509>
- Albert, R. and A.-L. Barabási (2002), Statistical mechanics of complex networks, *Rev. Mod. Phys.*, **74**, pp. 47–97, doi: 10.1103/RevModPhys.74.47. <https://link.aps.org/doi/10.1103/RevModPhys.74.47>
- Newman, M. (2003), The Structure and Function of Complex Networks, *SIAM Review*, **45**, 2, pp. 167–256, doi: 10.1137/S003614450342480
- Boccaletti, S. and V. Latora (2006), The Structure and Dynamics of Complex Networks



- Wang, X. F. and G. Chen (2003), Complex networks: small-world, scale-free and beyond, *IEEE Circuits and Systems Magazine*, **3**, 1, pp. 6–20, ISSN 1531-636X, doi: 10.1109/MCAS.2003.1228503
- Watts, D. and S. H. Strogatz (1998), Collective Dynamics of Small World Networks, *Nature*, **393**, pp. 440–2, doi: 10.1038/30918

2.2.5.5 Differential Equations

Differential equations are frequently used to model the physical phenomena of CPS and in particular the plant of CPSs. They are mathematical equations, consisting of derivatives and functions. Differential equations can be classified into ordinary differential equations (ODE), which have a single independent variable, differential-algebraic equations (DAE), which make use of algebraic equations and partial differential equations (PDE), which involve multi-variable functions and partial derivatives.

Implementing Languages

Differential Equations is a formalism for the following languages:

- Zélus (see section 2.3.76)
- Scilab Language (see section 2.3.54)
- XPPAUT Language (see section 2.3.75)
- PyDSTool Language (see section 2.3.50)
- Simulink Language (see section 2.3.59)
- Mathematica Language (see section 2.3.30)
- Modelica (see section 2.3.32)

References

- Winkel, B. (2016), Nagy, G. 2015. Ordinary Differential Equations. <https://www.simiode.org/resources/2681>

2.2.5.6 Discontinuous Systems

Discontinuous or non-smooth systems are a subclass of hybrid systems. There are several types, depending on the type of discontinuity in the model representing the system. Two main classes of discontinuous systems are observable: switching or variable structure systems, and jump or reset systems. Switching or variable structure systems are typically studied through the formalism of sliding motions (dynamics on the discontinuity surface) or sliding-mode-based control systems.

Implementing Languages

Discontinuous Systems is a formalism for the following languages:

- Stateflow Language (see section 2.3.64)
- Scilab Language (see section 2.3.54)
- XPPAUT Language (see section 2.3.75)
- PyDSTool Language (see section 2.3.50)
- Simulink Language (see section 2.3.59)
- Ptolemy Language (see section 2.3.49)
- Modelica (see section 2.3.32)



References

- I. Utkin, V. (1992), *Sliding Modes in Control Optimization*, pp. 12–28, ISBN 978-3-642-84381-5, doi: 10.1007/978-3-642-84379-2_2
- Navarro-Lopez, E. M. (2009a), Hybrid-automaton models for simulating systems with sliding motion: still a challenge, *IFAC Proceedings Volumes*, **42**, 17, pp. 322 – 327, ISSN 1474-6670, doi: <https://doi.org/10.3182/20090916-3-ES-3003.00056>, 3rd IFAC Conference on Analysis and Design of Hybrid Systems.
<http://www.sciencedirect.com/science/article/pii/S1474667015307825>
- Filippov, A. E. (1990), Differential Equations with Discontinuous Righthand Sides, *SIAM Review*, **32**, 2, pp. 312–315, doi: 10.1137/1032060.
<https://doi.org/10.1137/1032060>
- Navarro-Lopez, E. M. (2009b), Hybrid modelling of a discontinuous dynamical system including switching control, *IFAC Proceedings Volumes*, **42**, 7, pp. 87 – 92, ISSN 1474-6670, doi: <https://doi.org/10.3182/20090622-3-UK-3004.00019>, 2nd IFAC Conference on Analysis and Control of Chaotic Systems.
<http://www.sciencedirect.com/science/article/pii/S1474667015367458>

2.2.5.7 Discrete Event

The discrete event (DE) paradigm provides a representation of systems based on piecewise constant state variables. Contrarily to continuous systems, where variables change continuously in time, DE systems can only change at a finite number of instants during a finite time interval. The actions causing these changes are commonly called by events. Contrarily to discrete time models, where changes are periodic, in DE systems changes can occur randomly, i.e., without any following any periodicity.

Several strategies for organising DE systems have been developed. These descriptions are commonly termed by *world views* and provide different organisations for modelling DE systems. Common world views include event scanning Kiviat (1968), activity scanning Buxton and Laski (1962), and process interaction Dahl and Nygaard (1966).

The DEVS formalism Zeigler (1976), provides a formal description of modular DE systems, abstracting and unifying many of the concepts required for characterising DE systems. The co-simulation of DE systems was introduced in Zeigler (1984) that has established the separation between models and their simulators, enabling the interoperability of hierarchical and modular DE models.

Implementing Languages

Discrete Event is a formalism for the following languages:

- SIMUL8 Language (see section 2.3.58)
- Simio Language (see section 2.3.57)
- ARENA Language (see section 2.3.7)
- OMNet++ Language (see section 2.3.39)
- MS4 Me Language (see section 2.3.33)
- SimEvents Language (see section 2.3.56)
- FlexSim Language (see section 2.3.19)

References



- Kiviat, P. J. (1968), Introduction to the SIMSCRIPT II Programming Language, in *Proceedings of the Second Conference on Applications of Simulations*, Winter Simulation Conference, pp. 32–36.
<http://dl.acm.org/citation.cfm?id=800166.805259>
- Buxton, J. and J. Laski (1962), Control and Simulation Language, *Computer Journal*, , 5, pp. 194–199
- Zeigler, B. (1984), *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, San Diego
- Zeigler, B. (1976), *Theory of Modelling and Simulation*, Wiley

2.2.5.8 Electrical Linear Networks

Electrical linear networks (ELN) consist of linear electrical components that are interconnected to form a circuit. The electrical components considered are linear, such as: voltage/current sources, resistors, capacitors, inductors and transmission lines. Circuits falling within the ELN category are analysed with well-known linear methods, typically, frequency-domain methods.

Implementing Languages

Electrical Linear Networks is a formalism for the following languages:

- SystemC (see section 2.3.67)
- XPPAUT Language (see section 2.3.75)
- PyDSTool Language (see section 2.3.50)
- Simulink Language (see section 2.3.59)
- SPICE Language (see section 2.3.63)

References

- Banerjee, A. and B. Sur (2014), *Electrical Linear Networks in Practice and Theory*, Springer International Publishing, Cham, pp. 411–426, ISBN 978-3-319-01147-9, doi: 10.1007/978-3-319-01147-9_15.
https://doi.org/10.1007/978-3-319-01147-9_15

2.2.5.9 Entity Relationship

The entity-relationship model (ER) consists of entity types with relationships that can exist between instances of those types. Originating from the databases domain, ER defines information structures which can be implemented in a database. Some enhanced ER formalisms also include generalization-specialization relationships so that it can be used for the specification of domain ontologies.

Implementing Languages

Entity Relationship is a formalism for the following languages:

- UML (see section 2.3.69)
- AADL (see section 2.3.3)
- Entity Relationship Diagram (see section 2.3.16)
- AUTOSAR Language (see section 2.3.10)
- MetaH (see section 2.3.31)
- MARTE (see section 2.3.29)



References

- Chen, P. P.-S. (2001), *The Entity Relationship Model — Toward a Unified View of Data*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 205–234, ISBN 978-3-642-48354-7, doi: 10.1007/978-3-642-48354-7_7.
https://doi.org/10.1007/978-3-642-48354-7_7
- (2018), Entity Relationship Website, <https://www.smartdraw.com/entity-relationship-diagram/>

2.2.5.10 Fault Trees

Fault Trees is a formal model designed to analyze and evaluate the origin and the effects of faults in the components of an architecture at its subsystem or system level. The graphs represent the probability of component faults on the edges between nodes, allowing the quick calculation of risks by following the paths of the tree. Many variants have been proposed that extend the basic combinatorial nature of this formalism to include time, repairable components or repairing actions.

Implementing Languages

None

References

- Ruijters, E. and M. Stoelinga (2015), Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools, *Computer Science Review*, **15-16**, pp. 29 – 62, ISSN 1574-0137, doi: <https://doi.org/10.1016/j.cosrev.2015.03.001>.
<http://www.sciencedirect.com/science/article/pii/S1574013715000027>

2.2.5.11 Markov Chains

Markov chains are a probabilistic model of event sequences. When depicted as graph, each node in a Markov chain represents an event type (e.g. “rolling a die”) and the edges to other nodes represent the outcomes of the event (e.g. the probability of each side). The events need to support the Markov property. This means that predictions on the future evolution of the chain can be made based on the current state, and knowledge of previous evolutions has no influence on this prediction. Markov chains models can be built for discrete events as well as for continuous state space.

Implementing Languages

Markov Chains is a formalism for the following languages:

- PRISM Language (see section 2.3.44)

References

- Sigman, K. (2009), 1 IEOR 6711 : Continuous-Time Markov Chains
- Freedman, D. (1983), *Markov chains*, Springer-Verlag, New York-Berlin, ISBN 0-387-90808-0, corrected reprint of the 1971 original

2.2.5.12 Process Algebras

Process algebra refer to a group of different calculi that are used to study the evolution of concurrent processes and their relations, including communication, synchronisation and interactions. Individual processes are seen as agents that continuously interact with each other and the environment. Their well-defined semantics allow for the analysis and verification of highly complex systems. Common process algebras include the Calculus of Communicating Systems



(CCS), Communicating Sequential Processes (CSP) and the Algebra of Communicating Processes (ACP).

Implementing Languages

None

References

- van Glabbeek, R. J. (1987), Bounded nondeterminism and the approximation induction principle in process algebra, in *STACS 87*, Eds. F. J. Brandenburg, G. Vidal-Naquet and M. Wirsing, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 336–347, ISBN 978-3-540-47419-7
- Nicola, R. D. (2013), *A gentle introduction to Process Algebras*
- Fokkink, W. (2000), *Introduction to Process Algebra*, Springer-Verlag, Berlin, Heidelberg, 1st edition, ISBN 354066579X

2.2.5.13 TFPG (Timed Failure Propagation Graph)

A TFPG model is a relatively simple directed graph structure which identifies the paths along which failures are expected to propagate in the system. Nodes of a TFPG represent failure modes or discrepancies, and arcs represent failure propagation paths with a time interval representing the lower and upper bound of the failure propagation time. Logical operators AND and OR are used to represent logical combinations of failures to reach a mode and/or a discrepancy. TFPG can be used at design time to analyse faults propagation and their consequences. It can also be used at runtime to provide potential explanations to a fault signature that is observed during the system execution since consistency checking can be used to eliminate path on which timing constraints are not verified.

Implementing Languages

None

References

- Ofsthun, S. C. and S. Abdelwahed (2007), Practical applications of timed failure propagation graphs for vehicle diagnosis, in *2007 IEEE Autotestcon*, pp. 250–259, ISSN 1088-7725, doi: 10.1109/AUTEST.2007.4374226
- Misra, A. (1999), *Robust diagnostic system : structural redundancy approach*
- Bozzano, M., A. Cimatti, M. Gario and A. Micheli (2015), SMT-based Validation of Timed Failure Propagation Graphs, in *Proceedings of the Twenty-Ninth AAI Conference on Artificial Intelligence*, AAAI Press, AAAI'15, pp. 3724–3730, ISBN 0-262-51129-0.
<http://dl.acm.org/citation.cfm?id=2888116.2888233>

2.2.5.14 VDM (Vienna Development Method)

VDM is a formal description method for e.g. critical and concurrent systems, and compilers. It was developed in the 1970s and has seen significant industry usage. Its successor version VDM++ extends the modelling language to also support object-oriented systems.

Implementing Languages

None

References

- Bjørner, D. and C. B. Jones (Eds.) (1978), *The Vienna Development Method: The Meta-Language*, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-08766-4



2.3 Languages

The following subsections present the most commonly used languages for CPS development.

2.3.1 20-sim Block Diagrams

Block diagrams are a graphical programming and modelling language that is used to model within the 20-sim tool. They are similar to block diagrams in other languages and tools (e.g. Simulink).

Supported Formalisms

20-sim Block Diagrams is based on the following formalisms:

- Causal Block Diagrams (see section 2.2.5.3)

Supporting Tools

20-sim Block Diagrams is implemented by the following tools:

- 20-sim (see section 2.4.1)

References

- (2018), 20-sim Website, <http://20sim.com/>

2.3.2 20-sim Bond Graph

Bond graphs model idealized physical processes as graphs and graph networks. The language is implemented in 20-sim and allows the initial modelling of a-causal systems and performing causality analyses.

Supported Formalisms

20-sim Bond Graph is based on the following formalisms:

- Bond Graphs (see section 2.2.5.2)

Supporting Tools

20-sim Bond Graph is implemented by the following tools:

- 20-sim (see section 2.4.1)

References

- (2018), 20-sim Website, <http://20sim.com/>

2.3.3 AADL (Architecture Analysis and Design Language)

The Architecture Design & Analysis Language (AADL) is an architecture description language for the modeling of safety-critical real-time embedded systems. It allows the specification, analysis and code generation of these systems. It is standardized SAE as the AS5506C aerospace standard.

AADL allows modeling both the hardware and software parts of a system thus taking into account the deployment of a software application onto an execution platform. Many analysis and code generation tool are provided for AADL.

AADL is extensible and several annex languages have been proposed as described below:

Supported Formalisms

AADL is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)



- Data Flow (see section 2.2.2.1)
- Abstract State Machines (see section 2.2.1.1)

Supporting Tools

AADL is implemented by the following tools:

- AADL Inspector (see section 2.4.2)
- STOOD (see section 2.4.71)
- OSATE (see section 2.4.43)
- Ocarina (see section 2.4.39)
- MASIW (see section 2.4.30)
- Polychrony (see section 2.4.49)

References

- (2018b), AADL Website, <http://www.aadl.info/>
- (2018c), SAE AADL Specification, <http://standards.sae.org/as5506b/>

2.3.4 ACME

While most Architectural Description Languages (ADLs) considerably overlap on their core features, each ADL focuses on different aspects of the software architecture and problem categories. The creation of a global ADL would be impractical and developing mappings among each pair of languages would require an excessive amount of effort. Nevertheless, this diversity raises difficulties in transferring information among different ADLs. Acme was proposed as common representation of architectural concepts and to support the interchange of information using a generic language. This reduces the number of required transformations to those to and from ACME.

Language Features:

- an architectural ontology consisting of seven basic architectural design elements;
- a flexible annotation mechanism supporting association of non-structural information using externally defined sublanguages;
- a type mechanism for abstracting common, reusable architectural idioms and styles; and
- an open semantic framework for reasoning about architectural descriptions.

Supported Formalisms

None

Supporting Tools

ACME is implemented by the following tools:

- ACME Studio (see section 2.4.3)

References

- Garlan, D., R. T. Monroe and D. Wile (2000), Acme: Architectural description of component-based systems, in *Foundations of component-based systems*, Eds. G. T. Leavens and M. Sitaraman, pp. 47–68
- Wile, D. (1996), Semantics for the Architecture Interchange Language, ACME, in *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on*



SIGSOFT '96 Workshops, ACM, New York, NY, USA, ISAW '96, pp. 28–30, ISBN 0-89791-867-3, doi: 10.1145/243327.243341.

<http://doi.acm.org/10.1145/243327.243341>

- Garlan, D., R. Monroe and D. Wile (1997), Acme: An architecture description interchange language, in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, p. 7

2.3.5 AF3 Language (AutoFOCUS 3 Language)

This is the set of input languages of the AF3 tool.

Supported Formalisms

None

Supporting Tools

AF3 Language is implemented by the following tools:

- AF3 (see section 2.4.4)

References

- (2018), AF3 Website, <http://af3.fortiss.org/>

2.3.6 Alloy

Alloy is a specification language that allows the description of relations within a model. It is declarative in nature and allows checking of systems for correctness. It is accompanied by a tool implementation, the Alloy Analyzer.

Supported Formalisms

None

Supporting Tools

Alloy is implemented by the following tools:

- Alloy Analyzer (see section 2.4.5)

References

- (2018), Alloy Website, <http://alloy.lcs.mit.edu/alloy/>

2.3.7 ARENA Language

This is the input language of the ARENA simulation tool for discrete events.

Supported Formalisms

ARENA Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

ARENA Language is implemented by the following tools:

- ARENA (see section 2.4.9)

References

- (2018), Arena Website, <https://www.arenasimulation.com/>



2.3.8 AsmetaL

AsmetaL is the language used for the creation of models in the Asmeta toolset.

Supported Formalisms

AsmetaL is based on the following formalisms:

- Abstract State Machines (see section 2.2.1.1)

Supporting Tools

AsmetaL is implemented by the following tools:

- Asmeta Tools (see section 2.4.10)

References

- (2018a), Asmeta Website, <http://asmeta.sourceforge.net/>

2.3.9 AsmL (Abstract State Machine Language)

AsmL is a language implementation of the Abstract State Machine (ASM) formalism. It runs on the .NET platform and allows modelling, coding and testing.

Supported Formalisms

AsmL is based on the following formalisms:

- Abstract State Machines (see section 2.2.1.1)

Supporting Tools

None

References

- (2018c), AsmL Website, <https://www.microsoft.com/en-us/research/project/asml-abstract-state-machine-language/>

2.3.10 AUTOSAR Language (AUTomotive Open System ARchitecture)

AUTOSAR is a partnership of automotive stakeholders (vehicle manufacturers, suppliers, service providers and companies) working on establishing an open and standardized software architecture for automotive electronic control units (ECUs). AUTOSAR includes an architecture description language for the design of vehicular systems. Application software components are linked through abstract virtual function buses, which represent all hardware and system services offered by a vehicular system. This allows designers to focus on the application instead of the infrastructure software.

Supported Formalisms

AUTOSAR Language is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)

Supporting Tools

AUTOSAR Language is implemented by the following tools:

- System Desk (see section 2.4.73)

References

- (2018), AUTOSAR Website, <https://www.autosar.org/>



2.3.11 CCSL (Clock Constraint Specification Language)

The CCSL is a language that introduces the modelling of relations between clocks into the MARTE UML Profile.

Supported Formalisms

None

Supporting Tools

CCSL is implemented by the following tools:

- Time Square (see section 2.4.76)

References

- André, C. and F. Mallet (2008), Clock Constraints in UML/MARTE CCSL, Research Report RR-6540, INRIA.
<https://hal.inria.fr/inria-00280941>

2.3.12 CDL (Context Description Language)

CDL is a language that uses UML Activity and Sequence diagrams to model a system's context and environment. It is implemented in OBP toolset, which allows properties to be checked.

Supported Formalisms

None

Supporting Tools

CDL is implemented by the following tools:

- OBP Explorer (see section 2.4.38)
- TINA (see section 2.4.77)

References

- Dhaussy, P., J. Auvray, S. de Belloy, F. Boniol and E. Landel (2008), Using context descriptions and property definition patterns for software formal verification, in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pp. 89–96, doi: 10.1109/ICSTW.2008.52
- (2018), OBPCDL Website, <http://www.obpcdl.org/>

2.3.13 Cytoscape Language

This is the input language of the Cytoscape tool for biological networks.

Supported Formalisms

Cytoscape Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Cytoscape Language is implemented by the following tools:

- Cytoscape (see section 2.4.16)

References

- (2018), Cytoscape Website, <https://cytoscape.org/>



2.3.14 DEECo (Dependable Emergent Ensembles of Component)

DEECo is an architecture description language with a computational model that is used to design autonomous components. It is part of ASCENS project. More specifically, it captures the self-adaptation in the components and implicit exchanging of information in a dynamically formed group of components (i.e. ensemble). The formation of the ensemble is impacted by changes in the environment. The concepts presented in this computational model follows SCEL (see section 2.3.53) That is based on (see section 2.3.53).

From implementation perspective, the user uses java and follows a specific structure in writing the components and the ensembles, which conceptually matches the DSL presented in the papers. Multiple java annotations are employed to capture the structure.

Supported Formalisms

None

Supporting Tools

DEECo is implemented by the following tools:

- jDEECo (see section 2.4.26)

References

- Bures, T., I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit and F. Plasil (2013), DEECo: An Ensemble-based Component System, in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ACM, New York, NY, USA, CBSE '13, pp. 81–90, ISBN 978-1-4503-2122-8, doi: 10.1145/2465449.2465462.
<http://doi.acm.org/10.1145/2465449.2465462>
- Keznikl, J., T. Bures, F. Plasil and M. Kit (2012), Towards Dependable Emergent Ensembles of Components: The DEECo Component Model, in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pp. 249–252, doi: 10.1109/WICSA-ECSA.212.39
-
-

2.3.15 EAST-ADL

EAST-ADL is an ADL for the automotive embedded domain. It ties concepts from UML, SysML and AADL to the AUTOSAR environment.

Supported Formalisms

None

Supporting Tools

EAST-ADL is implemented by the following tools:

- Papyrus (see section 2.4.47)

References

- (2018), EAST-ADL Website, <http://www.east-adl.info>

2.3.16 Entity Relationship Diagram

Entity relationship diagrams are a visual, graphical modelling language for specifying relations within a domain. The language is primarily used to create models of relational databases.

Supported Formalisms



Entity Relationship Diagram is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)

Supporting Tools

None

References

- (2018), Entity Relationship Website, <https://www.smartdraw.com/entity-relationship-diagram/>

2.3.17 FCM (Flex-eWare Component Model)

FCM is an extension of the CORBA Component Model (CCM) in the aim of providing a common meta-model for component-oriented software modelling. The targeted domain of FCM modes are embedded and real-time systems. FCM notably offers flexible interaction through connectors and user defined port-types.

Supported Formalisms

None

Supporting Tools

None

References

- (2018), FCM Website, <http://www.ec3m.net/>
- Jan, M., C. Jouvray, F. Kordon, A. Kung, J. Lalande, F. Loiret, J. Navas, L. Pautet, J. Pulou, A. Radermacher and L. Seinturier (2012), Flex-eWare: a Flexible MDE-based Solution for Designing and Implementing Embedded Distributed Systems, *Software: Practice and Experience*, **42**, 12, pp. 1467–1494, doi: 10.1002/spe.1143. <https://hal.inria.fr/inria-00628310>

2.3.18 FIACRE (Intermediate Format for the Embedded Distributed Component Architectures)

FIACRE is a formally defined language for representing compositionally of component architectures. It supports the expression of embedded and distributed systems aspects (behaviour, timing) for formal verification and simulation.

Supported Formalisms

FIACRE is based on the following formalisms:

- Petri Net (see section 2.2.4.2)
- Timed Petri Nets (see section 2.2.4.5)

Supporting Tools

FIACRE is implemented by the following tools:

- OBP Explorer (see section 2.4.38)
- TINA (see section 2.4.77)

References

- (2018), FIACRE Website, <http://projects.laas.fr/fiacre/>



2.3.19 FlexSim Language

This is the input language of the FlexSim tool for discrete events simulation.

Supported Formalisms

FlexSim Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

FlexSim Language is implemented by the following tools:

- FlexSim (see section 2.4.20)

References

- (2018), Flexsim Website, <https://www.flexsim.com/flexsim>

2.3.20 FUMML (Foundational Subset for Executable UML Models)

FUMML is a subset of UML 2 that allows the structural and behavioural semantic description of systems. The models are analysable and executable.

Supported Formalisms

None

Supporting Tools

None

References

- (2018), FUMML Website, <https://www.omg.org/spec/FUMML/>

2.3.21 Gofer

Gofer is functional programming language that is a subset of Haskell and used to specify abstract state machines. It is used as input specification for the AsmGofer tool.

Supported Formalisms

Gofer is based on the following formalisms:

- Abstract State Machines (see section 2.2.1.1)

Supporting Tools

Gofer is implemented by the following tools:

- AsmGofer (see section 2.4.11)

References

- (2018b), AsmGofer Website, <https://tydo.eu/AsmGofer/>

2.3.22 HyTech Language

This is the input language of the HyTech model checker.

Supported Formalisms

HyTech Language is based on the following formalisms:

- Timed Automata (see section 2.2.1.11)

Supporting Tools



HyTech Language is implemented by the following tools:

- HyTech (see section 2.4.23)

References

- Henzinger, T. A., P.-H. Ho and H. Wong-Toi (1997), HyTech: A model checker for hybrid systems, in *Computer Aided Verification*, Ed. O. Grumberg, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 460–463, ISBN 978-3-540-69195-2
- (2018), HyTech Website, <https://ptolemy.berkeley.edu/projects/embedded/research/hytech/>

2.3.23 iGraph Language

This is the input language of the iGraph tool for complex networks analysis.

Supported Formalisms

iGraph Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

iGraph Language is implemented by the following tools:

- iGraph (see section 2.4.24)

References

- (2018), iGraph Website, <https://igraph.org/>

2.3.24 IRM (Invariant Refinement Method)

IRM provides the developer with a way to design Cyber-Physical Systems (CPS). The designed system follows the concepts presented in SCEL, which includes systems consisting of ensembles of components (e.g. DEECo-based systems). It is a method to map invariants from requirements level with parts of the system architecture.

Supported Formalisms

None

Supporting Tools

None

References

- (2018), IRM Website, <http://d3s.mff.cuni.cz/software/irm/>

2.3.25 IRM SA Language

IRM-SA (IRM for Self-Adaptation), which is an extension to IRM, provides developers with the ability to express alternative decompositions, which become as a part of self-adaptation during the runtime. The alternatives are associated with a set of assumptions related to the environment, so the choice of the alternatives should preserve the invariants under their associated assumptions.

Supported Formalisms

None

Supporting Tools

IRM SA Language is implemented by the following tools:



- IRM SA (see section 2.4.25)

References

- (2018), IRM Website, <http://d3s.mff.cuni.cz/software/irm/>

2.3.26 Kronos Language

This is the input language of the Kronos tool for timed automata.

Supported Formalisms

Kronos Language is based on the following formalisms:

- Linear Temporal Logic (see section 2.2.3.3)
- Timed Automata (see section 2.2.1.11)

Supporting Tools

Kronos Language is implemented by the following tools:

- Kronos (see section 2.4.27)

References

- Bérard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen and P. Mckenzie (2001), *KRONOS — Model Checking of Real-time Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 161–168, ISBN 978-3-662-04558-9, doi: 10.1007/978-3-662-04558-9_16.
https://doi.org/10.1007/978-3-662-04558-9_16
- Bozga, M., C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine (1998), Kronos: A model-checking tool for real-time systems, in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Eds. A. P. Ravn and H. Rischel, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 298–302, ISBN 978-3-540-49792-9
- (2018), Kronos Website, <http://www-verimag.imag.fr/~tripakis/openkronos.html>

2.3.27 LabVIEW Language

This is the input language of the LabVIEW tool.

Supported Formalisms

LabVIEW Language is based on the following formalisms:

- Causal Block Diagrams (see section 2.2.5.3)

Supporting Tools

LabVIEW Language is implemented by the following tools:

- LabVIEW (see section 2.4.28)

References

- (2018), LabVIEW Website, <http://www.ni.com/en-us/shop/labview/labview-details.html>

2.3.28 Lustre

Lustre is a declarative, synchronous language. It provides a sound model of computation with predictable performance, properties verification methods (liveness, deadlock freedom).

Supported Formalisms



Lustre is based on the following formalisms:

- Data Flow (see section 2.2.2.1)

Supporting Tools

Lustre is implemented by the following tools:

- Scade Suite (see section 2.4.62)

References

- (2018), SCADÉ Website, <https://www.ansys.com/products/embedded-software/ansys-scade-suite>

2.3.29 MARTE (Modeling and Analysis of Real-Time and Embedded systems)

The UML profile for MARTE is an extension of UML that adds missing aspects of the real time and embedded systems domain. The profile supports the modelling and analysis of schedulability, performance and time.

Supported Formalisms

MARTE is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)

Supporting Tools

MARTE is implemented by the following tools:

- Papyrus (see section 2.4.47)

References

- (2018), MARTE Website, <http://www.omg.org/spec/MARTE/>

2.3.30 Mathematica Language

This is the input language of the Mathematica tool.

Supported Formalisms

Mathematica Language is based on the following formalisms:

- Differential Equations (see section 2.2.5.5)
- Complex Networks (see section 2.2.5.4)

Supporting Tools

Mathematica Language is implemented by the following tools:

- Mathematica (see section 2.4.33)

References

- (2018a), Mathematica-Complex-Networks Website, <http://www.wolfram.com/mathematica/>
- (2018b), Mathematica Website, <http://www.wolfram.com/mathematica/>

2.3.31 MetaH

MetaH is a language for the modeling of reliable, real-time multiprocessor avionics system architectures that was developed in 1988 at Honeywell. It is the ancestor of the AADL language. See section 2.3.3.



Supported Formalisms

MetaH is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)

Supporting Tools

None

References

- (2018), MetaH Website, <http://hop1.info/showlanguage.prx?exp=7769>

2.3.32 Modelica

Modelica is a multi-domain modelling language for complex systems modelling and simulation. Its tool-implementations are wide spread and offer support for many different engineering domains (e.g. hydraulic, electric and mechanical).

Supported Formalisms

Modelica is based on the following formalisms:

- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Differential Equations (see section 2.2.5.5)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Hybrid Automata (see section 2.2.1.4)
- Discontinuous Systems (see section 2.2.5.6)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

Modelica is implemented by the following tools:

- Dymola (see section 2.4.17)
- AMESim (see section 2.4.7)
- Open Modelica (see section 2.4.41)

References

- (2018), MODELICA Website, <https://www.modelica.org/>

2.3.33 MS4 Me Language

Supported Formalisms

MS4 Me Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

MS4 Me Language is implemented by the following tools:

- MS4 Me (see section 2.4.34)

References

- (2018), MS4 Website, <http://www.ms4systems.com/>



2.3.34 Network Workbench Language

This is the input language of the Network Workbench tool for complex networks.

Supported Formalisms

Network Workbench Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Network Workbench Language is implemented by the following tools:

- Network Workbench (see section 2.4.35)

References

- (2018a), NetworkWorkbench Website, <http://nwb.cns.iu.edu/>

2.3.35 NetworkX Language

This is the input language of the NetworkX tool for complex networks.

Supported Formalisms

NetworkX Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

NetworkX Language is implemented by the following tools:

- NetworkX (see section 2.4.36)

References

- (2018b), NetworkX Website, <https://networkx.github.io/>

2.3.36 NuSMV Language

NuSMV is a symbolic model checker for the verification of industrial designs. It has its own input specification language (the NuSMV Language) that allows specifying the designs to be verified. The language syntax and semantics are described in the nuSMV user manual.

Supported Formalisms

NuSMV Language is based on the following formalisms:

- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Hybrid Automata (see section 2.2.1.4)
- CTL (see section 2.2.3.1)
- Temporal Logic (see section 2.2.3.4)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

NuSMV Language is implemented by the following tools:

- NuSMV (see section 2.4.37)

References

- (2018), NuSMV Website, <http://nusmv.fbk.eu/>



2.3.37 OCL (Object Constraint Language)

OCL is a declarative constraint language for UML models. It is standardized by the Object Management Group (OMG). A version exists for being used with Ecore models of the Eclipse Modeling Framework (EMF).

Supported Formalisms

None

Supporting Tools

OCL is implemented by the following tools:

- Papyrus (see section 2.4.47)

References

- (2018), OCL Website, <https://www.omg.org/spec/OCL>

2.3.38 OMEGA2

OMEGA2 is an extension of UML for the formal specification and validation of real-time systems.

Supported Formalisms

None

Supporting Tools

None

References

- Ober, I. and I. Dragomir (2010), OMEGA2: A New Version of the Profile and the Tools, in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 373–378, doi: 10.1109/ICECCS.2010.41

2.3.39 OMNet++ Language

Supported Formalisms

OMNet++ Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

OMNet++ Language is implemented by the following tools:

- OMNeT++ (see section 2.4.40)

References

- (2018), OMNetPP Website, <https://www.omnetpp.org/>

2.3.40 Pajek Language

This is the input language of the Pajek tool for complex networks.

Supported Formalisms

Pajek Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools



Pajek Language is implemented by the following tools:

- Pajek (see section 2.4.45)

References

- (2018), Pajek Website, <http://mrvar.fdv.uni-lj.si/pajek/>

2.3.41 Palladio Component Model

The Palladio Component model builds the foundation of the Palladio software approach. Its modularity takes user interaction with the system, and control and data flows into account and thereby generates a dynamic, adaptable quality-of-service model for early performance, reliability, maintainability, and cost predictions for software architectures.

Supported Formalisms

None

Supporting Tools

Palladio Component Model is implemented by the following tools:

- Palladio (see section 2.4.46)

References

- (2018), Palladio Website, <https://www.palladio-simulator.com/home/>

2.3.42 Petri Net Diagram

A representation of Petri nets as bi-partite graphs. Places are drawn as circles, transitions as rectangles and pre- and post-conditions as annotated, directed arcs. Tokens are drawn as black dots for Place-Transition nets, high level Petri nets might use other encoding for tokens of different types.

Supported Formalisms

Petri Net Diagram is based on the following formalisms:

- Petri Net (see section 2.2.4.2)
- Petri Nets with Priority (see section 2.2.4.3)
- Stochastic Petri Nets (see section 2.2.4.4)
- High Level Petri Nets (see section 2.2.4.1)
- Timed Petri Nets (see section 2.2.4.5)

Supporting Tools

Petri Net Diagram is implemented by the following tools:

- StrataGEM (see section 2.4.72)
- Alpina (see section 2.4.6)

References

- Ghezzi, C., D. Mandrioli, S. Morasca and M. Pezze (1991), A unified high-level Petri net formalism for time-critical systems, *IEEE Transactions on Software Engineering*, **17**, 2, pp. 160–172, ISSN 0098-5589, doi: 10.1109/32.67597
- Chachkov, S. and D. Buchs (2001), From formal specifications to ready-to-use software components: the concurrent object oriented Petri net approach, in *Proceedings Second*



International Conference on Application of Concurrency to System Design, pp. 99–110, doi: 10.1109/CSD.2001.981768

- Seiger, R., C. Keller, F. Niebling and T. Schlegel (2015), Modelling complex and flexible processes for smart cyber-physical environments, *Journal of Computational Science*, **10**, pp. 137 – 148, ISSN 1877-7503, doi: <https://doi.org/10.1016/j.jocs.2014.07.001>.
<http://www.sciencedirect.com/science/article/pii/S1877750314000970>

2.3.43 PNML (Petri Net Markup Language)

PNML is an XML-based encoding of Petri nets and Petri net diagrams.

Supported Formalisms

PNML is based on the following formalisms:

- Petri Net (see section 2.2.4.2)
- Petri Nets with Priority (see section 2.2.4.3)
- Stochastic Petri Nets (see section 2.2.4.4)
- Timed Petri Nets (see section 2.2.4.5)
- High Level Petri Nets (see section 2.2.4.1)

Supporting Tools

PNML is implemented by the following tools:

- StrataGEM (see section 2.4.72)
- ePNK (see section 2.4.18)

References

- (2018), PNML Website, <http://www.pnml.org/>

2.3.44 PRISM Language

The PRISM input language allows the specification of models within the PRISM tool, such as discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs) and probabilistic timed automata (PTAs).

Supported Formalisms

PRISM Language is based on the following formalisms:

- Markov Chains (see section 2.2.5.11)
- PTAs (see section 2.2.1.8)

Supporting Tools

PRISM Language is implemented by the following tools:

- PRISM (see section 2.4.51)

References

- (2018), PRISM Website, <http://www.prismmodelchecker.org>

2.3.45 Promela

Promela is a language for the modelling of distributed systems and concurrent processes. The language supports expression of synchronisation points and communication means (both syn-



chronous and asynchronous). Promela models can be analysed with the SPIN model checker, although many other tools also offer support for Promela models.

Supported Formalisms

None

Supporting Tools

Promela is implemented by the following tools:

- Spin (see section 2.4.69)

References

- Neumann, R. (2014), Using Promela in a Fully Verified Executable LTL Model Checker, in *Verified Software: Theories, Tools and Experiments*, Eds. D. Giannakopoulou and D. Kroening, Springer International Publishing, Cham, pp. 105–114, ISBN 978-3-319-12154-3

2.3.46 PROMISE Language (simPle ROBOT Mission SpECification Language)

PROMISE is a Domain Specific Language (DSL) that supports non-technical end users, i.e. users that do not necessarily have technical knowledge in mission specification. The DSL supports the specification of complex missions through tasks and operators. Tasks allow the specification of high-level actions that can be performed by a single robot. Specifically, tasks are specified by exploiting the PsALM specification patterns for robotic missions. Operators allow the composition of these tasks into complex missions.

Supported Formalisms

PROMISE Language is based on the following formalisms:

- Linear Temporal Logic (see section 2.2.3.3)

Supporting Tools

PROMISE Language is implemented by the following tools:

- PsALM (see section 2.4.53)
- PROMISE (see section 2.4.52)

References

- (2018), PROMISE Website, <https://sergiogarcia6.wixsite.com/promise>

2.3.47 PSC (Property Sequence Chart)

PSC is a simple but expressive language for specifying temporal properties. Two are the main requirements of PSC, simplicity and expressiveness. Remaining close to the graphical notation of UML2.0 Sequence Diagrams and Message Sequence Charts (MSC), the requirement of simplicity is satisfied. The PSC expressiveness is measured with the property specification patterns.

Supported Formalisms

PSC is based on the following formalisms:

- Linear Temporal Logic (see section 2.2.3.3)

Supporting Tools

PSC is implemented by the following tools:

- PragmaDevTracer (see section 2.4.50)



References

- (2018), PSC Website, <http://www.di.univaq.it/psc2ba/index.html>
- Autili, M., P. Inverardi and P. Pelliccione (2007), Graphical scenarios for specifying temporal properties: an automated approach, *Automated Software Engineering*, **14**, 3, pp. 293–340, ISSN 1573-7535, doi: 10.1007/s10515-007-0012-6.
<https://doi.org/10.1007/s10515-007-0012-6>

2.3.48 PSPF Language

PSPF provides a natural language interface based on a Structured English Grammar, an English-like specification scheme to capture pattern-based system properties, to specify temporal properties. It supports also the translation towards temporal logic via a collection of mappings from English to suitable logic formalisms, including LTL, CTL, MTL, TCTL, CSL, and PLTL.

Supported Formalisms

PSPF Language is based on the following formalisms:

- Temporal Logic (see section 2.2.3.4)

Supporting Tools

PSPF Language is implemented by the following tools:

- PSPWizard (see section 2.4.54)

References

- (2018b), PSPWizard Website, <http://www.ict.swin.edu.au/personal/mlumpe/PSPWizard/>

2.3.49 Ptolemy Language

This is the input language of the Ptolemy tool.

Supported Formalisms

Ptolemy Language is based on the following formalisms:

- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Hybrid Automata (see section 2.2.1.4)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Discontinuous Systems (see section 2.2.5.6)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

Ptolemy Language is implemented by the following tools:

- Ptolemy (see section 2.4.56)
- CyPhySim (see section 2.4.15)

References

- (2018), Ptolemy Website, <http://ptolemy.eecs.berkeley.edu/>

2.3.50 PyDSTool Language

This is the input language of the PyDSTool for the simulation and analysis of dynamical systems models of physical system.



Supported Formalisms

PyDSTool Language is based on the following formalisms:

- Electrical Linear Networks (see section 2.2.5.8)
- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Differential Equations (see section 2.2.5.5)
- Hybrid Automata (see section 2.2.1.4)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Discontinuous Systems (see section 2.2.5.6)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

PyDSTool Language is implemented by the following tools:

- PyDSTool (see section 2.4.57)

References

- (2018), PyDSTool Website, <http://www2.gsu.edu/~matrhc/PyDSTool.htm>

2.3.51 Random-Graph Networks

Supported Formalisms

Random-Graph Networks is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Random-Graph Networks is implemented by the following tools:

- NetworkX (see section 2.4.36)

References

2.3.52 Scale-Free Networks

Supported Formalisms

Scale-Free Networks is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Scale-Free Networks is implemented by the following tools:

- NetworkX (see section 2.4.36)

References

2.3.53 SCEL

SCEL is a specification language for dynamic distributed systems with collective behavior. It was represented in ASCENS project. More specifically, the components in the system exchange data implicitly (i.e. not by explicit calling of methods in other component), so the components are able to collaborate to achieve a common goal. This method increases the robustness in the system and fits to cases of continuously changing environments.

Supported Formalisms



None

Supporting Tools

None

References

- De Nicola, R., D. Latella, A. L. Lafuente, M. Loreti, A. Margheri, M. Massink, A. Morichetta, R. Pugliese, F. Tiezzi and A. Vandin (2015), *The SCEL Language: Design, Implementation, Verification*, Springer International Publishing, Cham, pp. 3–71, ISBN 978-3-319-16310-9, doi: 10.1007/978-3-319-16310-9_1.
https://doi.org/10.1007/978-3-319-16310-9_1
- (2018), ASCENS Website, <http://www.ascens-ist.eu/>

2.3.54 Scilab Language

This is the input language of the Scilab tool.

Supported Formalisms

Scilab Language is based on the following formalisms:

- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Differential Equations (see section 2.2.5.5)
- Hybrid Automata (see section 2.2.1.4)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Discontinuous Systems (see section 2.2.5.6)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

Scilab Language is implemented by the following tools:

- Scilab (see section 2.4.63)

References

- (2018), Scilab Website, <https://www.scilab.org/>

2.3.55 Signal

Signal is a synchronous language aiming at implementing applications with several clocks. Initially developed for signal processing applications, it was then studied in the context of real-time systems where functions are often executed with different activation rates. Several extensions of the language have been proposed since its first definition in 1982, including various extensions of synchronous data flow graphs.

Supported Formalisms

Signal is based on the following formalisms:

- Data Flow (see section 2.2.2.1)

Supporting Tools

Signal is implemented by the following tools:

- Polychrony (see section 2.4.49)

References



- (2018), Polychrony Website, <http://www.irisa.fr/espresso/Polychrony/>

2.3.56 SimEvents Language

This is the input language of the SimEvents simulation tool.

Supported Formalisms

SimEvents Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

SimEvents Language is implemented by the following tools:

- SimEvents (see section 2.4.64)

References

- (2018a), SimEvents Website, <https://www.mathworks.com/products/simevents.html>

2.3.57 Simio Language

This is the input language of the Simio simulation tool.

Supported Formalisms

Simio Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

Simio Language is implemented by the following tools:

- Simio (see section 2.4.65)

References

- (2018b), Simio Website, <https://www.simio.com/products/simulation-software.php>

2.3.58 SIMUL8 Language

Supported Formalisms

SIMUL8 Language is based on the following formalisms:

- Discrete Event (see section 2.2.5.7)

Supporting Tools

SIMUL8 Language is implemented by the following tools:

- SIMUL8 (see section 2.4.66)

References

- (2018), SIMUL8 Website, <https://www.simul8.com/>

2.3.59 Simulink Language

The Simulink tool provides a language that allows the modelling of multi-domain systems. The language has both graphical and textual features that model the signal flow between computation blocks.



Supported Formalisms

Simulink Language is based on the following formalisms:

- Electrical Linear Networks (see section 2.2.5.8)
- Differential Equations (see section 2.2.5.5)
- Causal Block Diagrams (see section 2.2.5.3)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Discontinuous Systems (see section 2.2.5.6)

Supporting Tools

Simulink Language is implemented by the following tools:

- Simulink (see section 2.4.67)

References

- (2018), Simulink Website, <https://ch.mathworks.com/de/products/simulink.html>

2.3.60 Small-World Networks

Supported Formalisms

Small-World Networks is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Small-World Networks is implemented by the following tools:

- NetworkX (see section 2.4.36)

References

2.3.61 SMT-LIB

SMT-LIB is a language designed to aid the development and exchange of Satisfiability Modulo Theories tools. SMT-LIB defines a language format for the description of theories and models.

Supported Formalisms

SMT-LIB is based on the following formalisms:

- First Order Logic (see section 2.2.3.2)

Supporting Tools

None

References

- (2018), SMT-LIB Website, <http://smtlib.cs.uiowa.edu/>

2.3.62 Spatio-Temporal UML Statechart

STUML statechart is an extension of UML MARTE statechart based on hybrid automata. Hybrid automata bring a set of differential equations into MARTE to represent the continuous dynamic behavior of the system.

Supported Formalisms

None



Supporting Tools

None

References

- Liu, Z., J. Liu, J. He and Z. Ding (2012), Spatio-temporal UML Statechart for Cyber-Physical Systems, in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pp. 137–146

2.3.63 SPICE Language (Simulation Program with Integrated Circuit Emphasis Language)

Supported Formalisms

SPICE Language is based on the following formalisms:

- Electrical Linear Networks (see section 2.2.5.8)

Supporting Tools

SPICE Language is implemented by the following tools:

- SPICE (see section 2.4.68)

References

2.3.64 Stateflow Language

This is the input language of the Mathworks Stateflow tool.

Supported Formalisms

Stateflow Language is based on the following formalisms:

- Non-Linear Hybrid Automata (see section 2.2.1.7)
- Hybrid Automata (see section 2.2.1.4)
- Stochastic Hybrid Automata (see section 2.2.1.9)
- Discontinuous Systems (see section 2.2.5.6)
- Linear Hybrid Automata (see section 2.2.1.6)

Supporting Tools

Stateflow Language is implemented by the following tools:

- Stateflow (see section 2.4.70)

References

- (2018), Stateflow Website, <https://www.mathworks.com/products/stateflow.html>

2.3.65 Stitch

Stitch is a language for describing strategies and tactics in self-adaptive systems at the architectural level. It allows representing the relation between decision trees and business objectives to be used in self-adaptation processes. It also provides the ability to express delays and uncertainties.

Supported Formalisms

None

Supporting Tools

Stitch is implemented by the following tools:



- Rainbow (see section 2.4.58)

References

- Cheng, S.-W. and D. Garlan (2012), Stitch: A language for architecture-based self-adaptation, *Journal of Systems and Software*, **85**, 12, pp. 2860 – 2875, ISSN 0164-1212, doi: <https://doi.org/10.1016/j.jss.2012.02.060>, self-Adaptive Systems.
<http://www.sciencedirect.com/science/article/pii/S0164121212000714>

2.3.66 SysML (Systems Modeling Language)

SysML is a modeling language for systems engineering supporting the specification, analysis, design, verification and validation of systems and systems-of-systems. It is an extension and adaptation of UML.. SysML simplifies UML by removing some of its software constructs, reuse 7 diagrams from UML and introduces two new diagrams for requirement engineering and for quantitative analysis (parametric diagram).

Supported Formalisms

None

Supporting Tools

SysML is implemented by the following tools:

- PTC MBSE (see section 2.4.55)
- Papyrus (see section 2.4.47)
- TTool (see section 2.4.78)

References

- (2018a), SysML Website, <http://sysml.org/>

2.3.67 SystemC

SystemC is an IEEE standardized Hardware Description Language (HDL) language. Contrary to other HDLs (e.g. VHDL, Verilog), SystemC is not a complete language by itself, but rather a set of C++ classes and macros, that allow the representation of HDL-concepts. SystemC includes built-in support for embedded concepts (e.g. mutex, semaphores, four-valued logic) and measures time in sub-second granularity (e.g. picosecond).

The use of C++ as a basis provides SystemC with flexibility and adaptability so that models are written just as any other C++ code, specifying ports, signals and channels. Functionality is modelled using methods, which execute at predefined events and threads, which run continuously until they finish or temporarily seize execution (SystemC provides a simulation kernel that prescribes cooperative multi-threading).

Most functional tooling and verification support focuses on the generation and verification of Transaction-Level Modelling and Register-Transfer Level designs, which is too low-level for large CPS purposes that focus on the combination of many components.

Supported Formalisms

SystemC is based on the following formalisms:

- Electrical Linear Networks (see section 2.2.5.8)

Supporting Tools

None

References



- Ruf, J., D. W. Hoffmann, J. Gerlach, T. Kropf, W. Rosenstiel and W. Müller (2001), The simulation semantics of systemC, in *DATE*
- (2018c), SystemC Website, <http://accelera.org/downloads/standards/systemc>
- Zhang, T., K. Chakrabarty and R. Fair (2002), *Microelectrofluidic Systems: Modeling and Simulation*, Nano- and Microscience, Engineering, Technology and Medicine, CRC Press, ISBN 978-1-4200-4049-4
- Herber, P. and S. Glesner (2015), Verification of Embedded Real-time Systems, in *Formal Modeling and Verification of Cyber-Physical Systems, 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015*, pp. 1–25, doi: 10.1007/978-3-658-09994-7_1.
https://doi.org/10.1007/978-3-658-09994-7_1
- Salem, A. (2003), Formal Semantics of Synchronous SystemC, in *Automation and Test in Europe Conference and Exhibition 2003 Design*, pp. 376–381, doi: 10.1109/DATE.2003.1253637
- Black, D. C., J. Donovan, B. Bunton and A. Keist (2010), *SystemC: From the Ground Up*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 0-387-29240-3

2.3.68 TEPE (Temporal Property Expression Language)

TEPE is a graphical Temporal Property Expression language based on SysML parametric diagrams. TEPE enriches the expressiveness of other common property languages in particular with the notion of physical time and unordered signal reception.

Supported Formalisms

None

Supporting Tools

TEPE is implemented by the following tools:

- TTool (see section 2.4.78)

References

- Knorreck, D., L. Apvrille and P. D. Saqui-Sannes (2010), TEPE: A SysML language for timed-constrained property modeling and formal verification, in *Proceedings of the UML&Formal Methods Workshop (UML&FM), November 2010*, pp. 17–47

2.3.69 UML (Unified Modeling Language)

UML is a set of graphical diagrams that allow the specification of a system's structure and behaviour. It is standardised by the Object Modeling Group (OMG). The language is flexible and can be adapted using UML Profiles.

Supported Formalisms

UML is based on the following formalisms:

- Entity Relationship (see section 2.2.5.9)

Supporting Tools

UML is implemented by the following tools:

- Papyrus (see section 2.4.47)
- TTool (see section 2.4.78)



- UML Analyzer (see section 2.4.79)

References

- (2018b), UML Website, <http://www.uml.org/>

2.3.70 UML-RT

UML-RT is an extension of UML for the modelling of real-time systems.

Supported Formalisms

None

Supporting Tools

None

References

- Grosu, R., M. Broy, B. Selic and G. Stefănescu (1999), *What is Behind UML-RT?*, Springer US, Boston, MA, pp. 75–90, ISBN 978-1-4615-5229-1, doi: 10.1007/978-1-4615-5229-1_6. https://doi.org/10.1007/978-1-4615-5229-1_6

2.3.71 UPPAAL Language

The UPPAAL language is based on three language references for model definition, requirements specification and an expression language. In combination, they can be used to simulate, analyse and verify extended timed automata.

Supported Formalisms

UPPAAL Language is based on the following formalisms:

- Timed Automata (see section 2.2.1.11)

Supporting Tools

UPPAAL Language is implemented by the following tools:

- UPPAAL (see section 2.4.80)

References

- (2018b), UPPAAL Website, <http://www.uppaal.com/>

2.3.72 UPPAAL SMC Language

This is the input language of UPPAAL SMC, which is a tool that allows the statistical model checking for probabilistic performance checking. The goal is to use statistical tools to give probabilities on whether a tool contains a property or not.

Supported Formalisms

UPPAAL SMC Language is based on the following formalisms:

- Stochastic Timed Automata (see section 2.2.1.10)

Supporting Tools

None

References

- (2018a), UPPAAL-SMC Website, <http://people.cs.aau.dk/~kg1/SSFT2015/SMC/>



2.3.73 VDM-SL (Vienna Development Method Specification Language)

VDM-SL is the formal specification language of the Vienna Development Method. It supports the description of data and functionality in textual form.

Supported Formalisms

None

Supporting Tools

VDM-SL is implemented by the following tools:

- Overture (see section 2.4.44)
- Crescendo (see section 2.4.14)

References

2.3.74 Visone Language

This is the input language of the Visone tool for social networks.

Supported Formalisms

Visone Language is based on the following formalisms:

- Complex Networks (see section 2.2.5.4)

Supporting Tools

Visone Language is implemented by the following tools:

- Visone (see section 2.4.81)

References

- (2018), Visone Website, <https://visone.info/>

2.3.75 XPPAUT Language

This is the input language of the XPPAUT tool for solving various mathematical equations.

Supported Formalisms

XPPAUT Language is based on the following formalisms:

- Electrical Linear Networks (see section 2.2.5.8)
- Differential Equations (see section 2.2.5.5)
- Discontinuous Systems (see section 2.2.5.6)

Supporting Tools

XPPAUT Language is implemented by the following tools:

- XPPAUT (see section 2.4.82)

References

- (2018), XPPAUT Website, <http://www.math.pitt.edu/~bard/xpp/xpp.html>

2.3.76 Zélus

Zélus is a Lustre-inspired, synchronous language. It aims to extend classical synchronous languages by adding support for ordinary differential equations. Thus it is possible to model continuous behaviour.

Supported Formalisms



Zélus is based on the following formalisms:

- Differential Equations (see section 2.2.5.5)
- Data Flow (see section 2.2.2.1)

Supporting Tools

None

References

- Bourke, T. and M. Pouzet (2013), Zélus: A Synchronous Language with ODEs, in *16th International Conference on Hybrid Systems: Computation and Control*
- Benveniste, A., T. Bourke, B. Caillaud and M. Pouzet (2011), Divide and recycle: types and compilation for a hybrid synchronous language, in *Proceedings of the ACM SIGPLAN/SIGBED 2011 conference on Languages, compilers, and tools for embedded systems, LCTES 2011*, Chicago, IL, United States, doi: 10.1145/1967677.1967687.
<https://hal.inria.fr/hal-00654112>

2.4 Tools

The following subsections present the most commonly used tools for CPS development.

2.4.1 20-sim

20-sim is a multi-domain modeling and simulation platform. It allows the specification of models using various formalisms (e.g. block diagrams, bond graphs, ODEs). 20-sim can be used for simulation, analysis and state space exploration of model designs. The tool further supports the generation of controller codes.

Supported Languages

20-sim is a tool for the following languages:

- 20-sim Block Diagrams (see section 2.3.1)
- 20-sim Bond Graph (see section 2.3.2)

References

- (2018), 20-sim Website, <http://20sim.com/>

2.4.2 AADL Inspector

AADL Inspector is an analysis tool for AADL. It can be used to perform static and dynamic analysis of AADL models by integrating other AADL-compliant tools thus making the integrated tools more usable. For instance, scheduling analysis, multi-agent simulation and code generation can be performed.

Supported Languages

AADL Inspector is a tool for the following languages:

- AADL (see section 2.3.3)

References

- (2018a), AADL-Inspector Website, <http://www.ellidiss.com/products/aadl-inspector/>



2.4.3 ACME Studio

ACME Studio is a graphical editor for models created in the Acme architecture description language. It is developed as a plugin for the Eclipse IDE and is adaptable, expandable and customizable using the Eclipse plugin features.

Supported Languages

ACME Studio is a tool for the following languages:

- ACME (see section 2.3.4)

References

- (2018), ACME-Studio Website, <http://www.cs.cmu.edu/~acme/AcmeStudio/>

2.4.4 AF3 (AutoFOCUS 3)

AF3 is an open-source modeling tool developed by Fortiss. AF3 drives the MDE approach by supporting requirements modelling, simulation, code generation, formal verification and similar types of modelling tasks.

Supported Languages

AF3 is a tool for the following languages:

- AF3 Language (see section 2.3.5)

References

- (2018), AF3 Website, <http://af3.fortiss.org/>

2.4.5 Alloy Analyzer

The Alloy Analyzer is a tool developed alongside the Alloy language. It can be used to find inconsistencies and check properties of the declarative Alloy models. Similar to many popular model checkers, it can generate counter examples and graphically visualise the model structures.

Supported Languages

Alloy Analyzer is a tool for the following languages:

- Alloy (see section 2.3.6)

References

- (2018), Alloy Website, <http://alloy.lcs.mit.edu/alloy/>

2.4.6 Alpina

Supported Languages

Alpina is a tool for the following languages:

- Petri Net Diagram (see section 2.3.42)

References

2.4.7 AMESim (Advanced Modeling Environment for Simulations)

AMESim stands for Advanced Modeling Environment for performing Simulations of engineering systems. It is based on an intuitive graphical interface in which the system is displayed throughout the simulation process.

Supported Languages



AMESim is a tool for the following languages:

- Modelica (see section 2.3.32)

References

- (2018), AMESim Website, <https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html>

2.4.8 Any Logic

AnyLogic is a multi-method modelling tool, supporting various modelling formalisms and languages, including statecharts, process flowcharts, and action charts. The tool allows the use of multiple simulation paradigms, including system dynamics, discrete events and agent-based models. Its graphical user interface and well-integrated tools and libraries allow rapid modelling processes in various domains, such as business processes, manufacturing and logistics.

Supported Languages

None

References

- (2018), Any-Logic Website, <http://www.anylogic.com/>

2.4.9 ARENA

Supported Languages

ARENA is a tool for the following languages:

- ARENA Language (see section 2.3.7)

References

- (2018), Arena Website, <https://www.arenasimulation.com/>

2.4.10 Asmeta Tools

Asmeta Tools is a collection of software utilities for interaction with Asmeta models. Asmeta Tools allow for example the editing, visualisation, simulation and model checking of Asmeta models. The tools are distributed as plugins for the Eclipse IDE.

Supported Languages

Asmeta Tools is a tool for the following languages:

- AsmetaL (see section 2.3.8)

References

- (2018a), Asmeta Website, <http://asmeta.sourceforge.net/>

2.4.11 AsmGofer

AsmGofer is a tool for Abstract State Machine (ASM) programming. It provides a modern ASM interpreter making use of the functional programming language Gofer.

Supported Languages

AsmGofer is a tool for the following languages:

- Gofer (see section 2.3.21)

References

- (2018b), AsmGofer Website, <https://tydo.eu/AsmGofer/>



2.4.12 Capella

Capella is a modeling tool for the ARCADIA method distributed as an Eclipse plugin. The basis of ARCADIA enforces three main modelling aspects, i.e. the requirements engineering, the architecture modelling and the need analysis.

Supported Languages

None

References

- (2018), Capella Website, <https://polarsys.org/capella/>

2.4.13 COMSOL Multiphysics

COMSOL Multiphysics is a tool that allows the simulation, solving and analysis of systems in various domains, such as electrical, mechanical and chemical applications.

Supported Languages

None

References

- (2018), COMSOL Website, <https://www.comsol.com/>

2.4.14 Crescendo

Crescendo is a tool built upon the Overture tool and 20-sim, to offer co-simulation, design and modelling capabilities for cyber-physical systems.

Supported Languages

Crescendo is a tool for the following languages:

- VDM-SL (see section 2.3.73)

References

- (2018), Crescendo Website, <http://crescendotool.org/>

2.4.15 CyPhySim

CyPhySim is a Cyber-Physical Simulator based on Ptolemy II.

Supported Languages

CyPhySim is a tool for the following languages:

- Ptolemy Language (see section 2.3.49)

References

- (2018), CyPhySim Website, <http://cyphysim.org/>

2.4.16 Cytoscape

Cytoscape is a tool for the visualisation of biological networks.

Supported Languages

Cytoscape is a tool for the following languages:

- Cytoscape Language (see section 2.3.13)

References

- (2018), Cytoscape Website, <https://cytoscape.org/>



2.4.17 Dymola

Dymola is a commercial implementation of the Modelica language. It allows modelling and simulation of complex multi-domain systems.

Supported Languages

Dymola is a tool for the following languages:

- Modelica (see section 2.3.32)

References

- (2018), Dymola Website, <https://www.3ds.com/products-services/catia/products/dymola/>

2.4.18 ePNK

The Eclipse Petri Net Kernel (ePNK) is an Eclipse-based tool that allows the easy creation of Petri net tools based on the PNML (Petri Net Markup Language) exchange format. It aims to facilitate the definition of various types of Petri nets and provide a mutual, generic graphical editor for all types of Petri nets.

Supported Languages

ePNK is a tool for the following languages:

- PNML (see section 2.3.43)

References

- (2018), ePNK Website, www.imm.dtu.dk/~ekki/projects/ePNK/

2.4.19 ESMoL (Embedded Systems Modeling Language)

ESMoL is a set of DSMLs for the modelling of embedded systems. The languages address various aspects of the safety-critical systems domain and focus on separation of concerns between control engineering, hardware specification, and software development teams. ESMoL is further used to specify relations between controllers, implementing functions and hardware platforms.

Supported Languages

None

References

- Porter, J., G. Hemingway, N. Kottenstette, G. Karsai and J. Sztipanovits (2011), The ESMoL Language and Tools for High-Confidence Distributed Control Systems Design . Part 1 : Design Language , Modeling Framework , and Analysis

2.4.20 FlexSim

Supported Languages

FlexSim is a tool for the following languages:

- FlexSim Language (see section 2.3.19)

References

- (2018), Flexsim Website, <https://www.flexsim.com/flexsim>



2.4.21 FlyAQ

FlyAQ is a tool that enables non-expert users to program missions of autonomous multi-copters. <http://www.flyaq.it> FlyAQ enables the specification of mission through a domain-specific modeling language, which can be effectively used by end-users with no technical expertise, e.g., firefighters and rescue workers. The FlyAQ generation method automatically derives the lower level behaviours that each drone must perform to accomplish the specified mission, prevents collisions between drones and obstacles, and ensures the preservation of no-fly zones.

Supported Languages

None

References

- Further details might be found at: <https://doi.org/10.1145/2976767.2976794>

2.4.22 GreatSPN

Supported Languages

None

References

2.4.23 HyTech

HyTech is a tool for the analysis hybrid systems that are specified as collections of automata with discrete and continuous components. It supports the verification of temporal requirements using symbolic model checking.

Supported Languages

HyTech is a tool for the following languages:

- HyTech Language (see section 2.3.22)

References

- Henzinger, T. A., P.-H. Ho and H. Wong-Toi (1997), HyTech: A model checker for hybrid systems, in *Computer Aided Verification*, Ed. O. Grumberg, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 460–463, ISBN 978-3-540-69195-2
- (2018), HyTech Website, <https://ptolemy.berkeley.edu/projects/embedded/research/hytech/>
- Alur, R., T. A. Henzinger and P.-H. Ho (1996), Automatic symbolic verification of embedded systems, *IEEE Transactions on Software Engineering*, **22**, 3, pp. 181–201, ISSN 0098-5589, doi: 10.1109/32.489079

2.4.24 iGraph

iGraph is a tool for complex networks typically used in the R Environment for Statistical Computing.

Supported Languages

iGraph is a tool for the following languages:

- iGraph Language (see section 2.3.23)

References



- (2018), iGraph Website, <https://igraph.org/>

2.4.25 IRM SA (Invariant Refinement Method for Self-Adaptation)

IRM-SA is a toolchain for creating IRM-SA models. The method is focused on the specification of ensemble-based component system (EBCS) - DEECo. It is developed using Eclipse EMF and GMF and the Epsilon modeling languages.

Supported Languages

IRM SA is a tool for the following languages:

- IRM SA Language (see section 2.3.25)

References

- (2018), IRM Website, <http://d3s.mff.cuni.cz/software/irm/>

2.4.26 jDEECo (Java Dependable Emergent Ensembles of Component)

JDEECo is a prototype tool that illustrates basic DEECo concepts. JDEECo provides a runtime environment for DEECo that is built using the Java language. With jDEECo, users can build their own use cases following the DEECo concepts.

Supported Languages

jDEECo is a tool for the following languages:

- DEECo (see section 2.3.14)

References

- (2018), JDEECo Website, <https://github.com/d3scomp/JDEECo/wiki>

2.4.27 Kronos

Kronos is a timed automata model checker, which uses the region graph method for verification. It supports the TCTL temporal logic.

Supported Languages

Kronos is a tool for the following languages:

- Kronos Language (see section 2.3.26)

References

- Bérard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen and P. Mckenzie (2001), *KRONOS — Model Checking of Real-time Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 161–168, ISBN 978-3-662-04558-9, doi: 10.1007/978-3-662-04558-9_16.
https://doi.org/10.1007/978-3-662-04558-9_16
- Bozga, M., C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine (1998), Kronos: A model-checking tool for real-time systems, in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Eds. A. P. Ravn and H. Rischel, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 298–302, ISBN 978-3-540-49792-9
- (2018), Kronos Website, <http://www-verimag.imag.fr/~tripakis/openkronos.html>



2.4.28 LabVIEW

LabVIEW is a platform for visual dataflow programming created by National Instruments. It features a visual programming language, simulation capabilities and supports hardware-in-the-loop system development.

Supported Languages

LabVIEW is a tool for the following languages:

- LabVIEW Language (see section 2.3.27)

References

- (2018), LabVIEW Website, <http://www.ni.com/en-us/shop/labview/labview-details.html>

2.4.29 Lola

Supported Languages

None

References

2.4.30 MASIW (Modular Avionics System Integrator Workplace)

MASIW is a tool to for the design, verification and code generation of real-time aviation electronics systems based on Integrated Modular Avionics architectures. MASIW makes use of AADL as specification language.

Supported Languages

MASIW is a tool for the following languages:

- AADL (see section 2.3.3)

References

- (2018a), MASIW Website, <http://www.ispras.ru/en/technologies/masiw/>

2.4.31 MAST

MAST is a suite of tools for the analysis of distributed systems. Its focus lies on the assessment of schedulability and timing requirements of real-time systems.

Supported Languages

None

References

- (2018b), MAST Website, <https://mast.unican.es/>

2.4.32 MAT Sim

MATSim (Multi-Agent Transport Simulation) is a framework that is used to simulate road traffic and replanning. It allows the developer to model agents in large scale systems.

Supported Languages

None

References

- (2018), MATSim Website, <https://matsim.org/>



- Horni, A., K. Nagel and K. Axhausen (Eds.) (2016), *Multi-Agent Transport Simulation MAT-Sim*, Ubiquity Press, London, ISBN 978-1-909188-75-4, 978-1-909188-76-1, 978-1-909188-77-8, 978-1-909188-78-5, doi: 10.5334/baw

2.4.33 Mathematica

Supported Languages

Mathematica is a tool for the following languages:

- Mathematica Language (see section 2.3.30)

References

- (2018a), Mathematica-Complex-Networks Website, <http://www.wolfram.com/mathematica/>
- (2018b), Mathematica Website, <http://www.wolfram.com/mathematica/>

2.4.34 MS4 Me

Supported Languages

MS4 Me is a tool for the following languages:

- MS4 Me Language (see section 2.3.33)

References

- (2018), MS4 Website, <http://www.ms4systems.com/>

2.4.35 Network Workbench

Network Workbench is an Interactive analysis and visualisation tool for large-scale networks.

Supported Languages

Network Workbench is a tool for the following languages:

- Network Workbench Language (see section 2.3.34)

References

- (2018a), NetworkWorkbench Website, <http://nwb.cns.iu.edu/>

2.4.36 NetworkX

NetworkX is a tool for Interactive analysis of complex networks in Python.

Supported Languages

NetworkX is a tool for the following languages:

- Small-World Networks (see section 2.3.60)
- NetworkX Language (see section 2.3.35)
- Scale-Free Networks (see section 2.3.52)
- Random-Graph Networks (see section 2.3.51)

References

- (2018b), NetworkX Website, <https://networkx.github.io/>



2.4.37 NuSMV

NuSMV is a popular symbolic model checker. It is employed in many other tools for systems and model analysis. It is further the basis of NuXMV, a model checker for synchronous systems.

Supported Languages

NuSMV is a tool for the following languages:

- NuSMV Language (see section 2.3.36)

References

- (2018), NuSMV Website, <http://nusmv.fbk.eu/>

2.4.38 OBP Explorer

The OBP toolset is an implementation of the OBP-CDL language. It allows model properties to be checked.

Supported Languages

OBP Explorer is a tool for the following languages:

- CDL (see section 2.3.12)
- FIACRE (see section 2.3.18)

References

- (2018), OBPCDL Website, <http://www.obpcdl.org/>

2.4.39 Ocarina

Ocarina is a tool for the processing of AADL models. It supports model and schedulability analysis, parsing, code generation and model checking. Ocarina can be used as stand-alone solution or integrated into OSATE and other tools.

Supported Languages

Ocarina is a tool for the following languages:

- AADL (see section 2.3.3)

References

- (2018), Ocarina Website, <http://www.openaadl.org/ocarina.html>

2.4.40 OMNeT++

OMNeT++ is a discrete event simulation library for network simulators, implemented in C++. This design choice renders the tool extensible and modular.

Supported Languages

OMNeT++ is a tool for the following languages:

- OMNet++ Language (see section 2.3.39)

References

- (2018), OMNetPP Website, <https://www.omnetpp.org/>

2.4.41 Open Modelica

Open Modelica is an open-source tool for the Modelica language. The popular tool offers modelling and simulation capabilities and sees industrial and academic usage.



Supported Languages

Open Modelica is a tool for the following languages:

- Modelica (see section 2.3.32)

References

- (2018), MODELICA Website, <https://www.modelica.org/>

2.4.42 ORIS

Supported Languages

None

References

2.4.43 OSATE (Open Source AADL Tool Environment)

OSATE is the reference tool for AADL. It allows editing AADL models using the textual and graphical syntax. It also supports several analyses such as resource scheduling, latency and response time and resources consumption. It also support most standard AADL annexes such as the Error model and behavior annexes and provides ALISA, a workbench for assurance case modeling including requirements management and verification activities.

Supported Languages

OSATE is a tool for the following languages:

- AADL (see section 2.3.3)

References

- (2018), OSATE Website, <http://osate.org/>

2.4.44 Overture

Overture is an Eclipse-based tool that notably supports the Vienna Development Method (VDM). It supports various VDM dialects (VDM-SL, VDM++, VDM-RT).

Supported Languages

Overture is a tool for the following languages:

- VDM-SL (see section 2.3.73)

References

- (2018), Overture Website, <https://www.overturetool.org/>

2.4.45 Pajek

Pajek is a tool for the analysis and visualisation of large-scale complex networks.

Supported Languages

Pajek is a tool for the following languages:

- Pajek Language (see section 2.3.40)

References

- (2018), Pajek Website, <http://mrvar.fdv.uni-lj.si/pajek/>



2.4.46 Palladio

Palladio is an architecture based analysis and simulation tool. It focuses on model-based discovery of performance bottlenecks, scalability issues, and reliability threats.

Supported Languages

Palladio is a tool for the following languages:

- Palladio Component Model (see section 2.3.41)

References

- (2018), Palladio Website, <https://www.palladio-simulator.com/home/>

2.4.47 Papyrus

Papyrus is an Eclipse-based UML, SysML and domain-specific modelling environment. The tool is customisable and expandable and supports UML profiles.

Supported Languages

Papyrus is a tool for the following languages:

- UML (see section 2.3.69)
- OCL (see section 2.3.37)
- EAST-ADL (see section 2.3.15)
- MARTE (see section 2.3.29)
- SysML (see section 2.3.66)

References

- (2018), Papyrus Website, <https://www.eclipse.org/papyrus/>

2.4.48 PHAVer (Polyhedral Hybrid Automaton Verifier)

PHAVer is a model checking and verification tool for hybrid automata. It allows the verification of safety properties and offers robust arithmetic, unlimited precision and support for compositional and assume-guarantee reasoning.

Supported Languages

None

References

- (2018), PHAVer Website, http://www-verimag.imag.fr/~frehse/phaver_web/

2.4.49 Polychrony

Polychrony was initially developed as an environment for the Signal language (see section 2.3.55). Polychrony is an integrated development environment including an editor and a compiler of Signal programs. More recently, Polychrony has been extended to support subsets of the AADL language 2.3.3.

Supported Languages

Polychrony is a tool for the following languages:

- AADL (see section 2.3.3)
- Signal (see section 2.3.55)



References

- (2018), Polychrony Website, <http://www.irisa.fr/espresso/Polychrony/>

2.4.50 PragmaDevTracer

PragmaDev Tracer is a free tool for checking that traces of the system execution match requirements expressed in PSC.

Supported Languages

PragmaDevTracer is a tool for the following languages:

- PSC (see section 2.3.47)

References

- <http://www.pragmadev.com/product/tracer.html>

2.4.51 PRISM

PRISM is a popular model checker for probabilistic models. It features the analysis of systems with random or probabilistic behaviour.

Supported Languages

PRISM is a tool for the following languages:

- PRISM Language (see section 2.3.44)

References

- (2018), PRISM Website, <http://www.prismmodelchecker.org>

2.4.52 PROMISE (simPle RObot Mission SpEcification)

PROMISE (i) provides the users with a user-friendly instrument to specify missions (without requiring them to know programming languages, robotics, and mathematics); (ii) enables a rigorous and precise specification so to enable the use of planners, analysis tools, simulators, or other modules; and (iii) is platform-independent and highly customizable. PROMISE is developed as a standalone application, and it could be integrated with various tools and robotic platforms

Supported Languages

PROMISE is a tool for the following languages:

- PROMISE Language (see section 2.3.46)

References

- (2018), PROMISE Website, <https://sergiogarcia6.wixsite.com/promise>

2.4.53 PsALM

PsALM is a tool for specifying missions for mobile robot platforms by composing reusable specification patterns. The core motivation behind the tool is that manually writing LTL and CTL specifications for robotic missions is often tedious, difficult, and error prone. PsALM helps practitioners to write specifications by providing a library of reusable specification patterns, each of which describes a particular intent for the robot (e.g., to visit a set of locations in a particular order), that can be composed to produce a complete description of the intended behaviours of the robot. The tool has been accepted as tool paper at ICSE 2019.

Supported Languages



None

References

- (2018), PsALM Website, <http://178.62.206.217/patterns/psalm/>

2.4.54 PSPWizard (Property Specification Pattern Wizard)

PSPWizard is designed to allow system engineers to flexibly specify properties, without knowledge in qualitative, real-time, or probabilistic logic. PSPWizard makes use of a catalogue of specification patterns described in Autili et al. (2015).

Supported Languages

PSPWizard is a tool for the following languages:

- PSPF Language (see section 2.3.48)

References

- (2018b), PSPWizard Website, <http://www.ict.swin.edu.au/personal/mlumpe/PSPWizard/>
- (2018a), Property Specification Patterns Website, <http://ps-patterns.wikidot.com>

2.4.55 PTC MBSE (PTC Model-based System Engineering)

PTC Model-based System Engineering is a set of tools supporting complex systems engineering. It consists of PTC Integrity Modeler, which is a modeling tool for SysML including variability modeling, PTC Integrity Process Director, which is a process engineering tool to manage any processes including engineering and software development processes, and PTC Integrity Asset Library for supporting software and systems assets reuse.

PTC's solution is issued from the former Atego Artisan Studio that was bought in 2014.

Supported Languages

PTC MBSE is a tool for the following languages:

- SysML (see section 2.3.66)

References

- (2018), PTC Website, <https://www.ptc.com/>

2.4.56 Ptolemy

Ptolemy II is a modelling platform, supporting various modelling paradigms, such as actor-oriented, real-time, discrete-event, continuous time. It supports the modelling, simulation and analysis of heterogeneous models, notably hybrid automata. It is further the platform for other tools built upon its models of computation.

Supported Languages

Ptolemy is a tool for the following languages:

- Ptolemy Language (see section 2.3.49)

References

- (2018), Ptolemy Website, <http://ptolemy.eecs.berkeley.edu/>



2.4.57 PyDSTool

PyDSTool is a simulation and analysis tool for dynamical systems models of physical systems. It supports ordinary differential equations, differential algebraic equations, maps and hybrid system.

Supported Languages

PyDSTool is a tool for the following languages:

- PyDSTool Language (see section 2.3.50)

References

- (2018), PyDSTool Website, <http://www2.gsu.edu/~matrhc/PyDSTool.htm>

2.4.58 Rainbow

Rainbow is a tool building self-adaptive systems based on MAPE-K method and using Stitch language. The tool allows the developer to describe strategies and tactics, in addition to capturing context in adaptation decisions.

Supported Languages

Rainbow is a tool for the following languages:

- Stitch (see section 2.3.65)

References

- (2018), Rainbow Website, <http://www.cs.cmu.edu/~able/research/rainbow/>

2.4.59 Rational Software Architect

IBM's Rational Software Architect is a tool suite for the end-to-end modelling and development of software systems. It features extensive design and modelling capabilities around various modelling formalisms and languages and further features extensive analysis and reporting capabilities.

Supported Languages

None

References

- (2018), RSA Website, <https://www.ibm.com/developerworks/downloads/r/architect/>

2.4.60 Remes

Remes is a state-machine based modelling language for embedded systems. It features the representation of resources (e.g. storage, computation, electrical power) and continuous time.

Supported Languages

None

References

- Seceleanu, C., A. Vulgarakis and P. Pettersson (2009), REMES: A resource model for embedded systems, pp. 84–94, doi: 10.1109/ICECCS.2009.49



2.4.61 ROS (Robot Operating System)

The Robot Operating System is a middleware for robotics systems. It provides libraries for the design, creation and analysis of embedded systems.

Supported Languages

None

References

- (2018), ROS Website, <http://www.ros.org/>

2.4.62 Scade Suite

SCADE is a tool suite for the modelling, simulation and analysis of mission and safety-critical embedded systems. Its focus lies on the software side, which can be modelled. SCADE applications involve requirements modelling, model design, verification and code-generation.

Supported Languages

Scade Suite is a tool for the following languages:

- Lustre (see section 2.3.28)

References

- (2018), SCADE Website, <https://www.ansys.com/products/embedded-software/ansys-scade-suite>

2.4.63 Scilab

Scilab is a tool for mathematics, statistics, signal processing and control systems.

Supported Languages

Scilab is a tool for the following languages:

- Scilab Language (see section 2.3.54)

References

- (2018), Scilab Website, <https://www.scilab.org/>

2.4.64 SimEvents

Supported Languages

SimEvents is a tool for the following languages:

- SimEvents Language (see section 2.3.56)

References

- (2018a), SimEvents Website, <https://www.mathworks.com/products/simevents.html>

2.4.65 Simio

Supported Languages

Simio is a tool for the following languages:

- Simio Language (see section 2.3.57)

References



- (2018b), Simio Website, <https://www.simio.com/products/simulation-software.php>

2.4.66 SIMUL8

Supported Languages

SIMUL8 is a tool for the following languages:

- SIMUL8 Language (see section 2.3.58)

References

- (2018), SIMUL8 Website, <https://www.simul8.com/>

2.4.67 Simulink

Simulink is a multi-domain simulation and modelling environment. Its main language is block diagram based, although it also supports a programming and scripting language for advanced, customised extension. Due to its flexibility and large object library it is popular and can be seen as a de-facto standard for systems modelling.

Supported Languages

Simulink is a tool for the following languages:

- Simulink Language (see section 2.3.59)

References

- (2018), Simulink Website, <https://ch.mathworks.com/de/products/simulink.html>

2.4.68 SPICE (Simulation Program with Integrated Circuit Emphasis)

Supported Languages

SPICE is a tool for the following languages:

- SPICE Language (see section 2.3.63)

References

- Nagel, L. W. and D. Pederson (1973), SPICE (Simulation Program with Integrated Circuit Emphasis), Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley.
<http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html>

2.4.69 Spin

Spin is wide-spread model verification tool. It is known for its verification capabilities of parallel and multi-threaded software applications.

Supported Languages

Spin is a tool for the following languages:

- Promela (see section 2.3.45)

References

- (2018), Spin Website, <http://spinroot.com/spin/whatispin.html>



2.4.70 Stateflow

Stateflow is a tool developed by MathWorks for control logic modeling of reactive systems via state machines and flow charts within a Simulink model.

Supported Languages

Stateflow is a tool for the following languages:

- Stateflow Language (see section 2.3.64)

References

- (2018), Stateflow Website, <https://www.mathworks.com/products/stateflow.html>

2.4.71 STOOD

STOOD is a tool suite for the system design and modelling. It complies with AADL and follows the Hierarchical Object Oriented Design (HOOD) method.

Supported Languages

STOOD is a tool for the following languages:

- AADL (see section 2.3.3)

References

- (2018), STOOD Website, <https://www.ellidiss.com/products/stood/>

2.4.72 StrataGEM (Strategy Generic Extensible Modelchecker)

(StrataGEM is a tool for the analysis and symbolic model-checking of concurrent models such as Petri nets. It is based on term rewriting and uses decision diagrams for efficiency.

Supported Languages

StrataGEM is a tool for the following languages:

- PNML (see section 2.3.43)
- Petri Net Diagram (see section 2.3.42)

References

- http://link.springer.com/chapter/10.1007/978-3-319-07734-5_20

2.4.73 System Desk

SystemDesk is an AUTOSAR tool aimed at the modelling of system architectures. It can generate virtual ECUs (V-ECUs) from application software, which can be tested using other dSPACE simulation platforms.

Supported Languages

System Desk is a tool for the following languages:

- AUTOSAR Language (see section 2.3.10)

References

- (2018b), System-Desk Website, https://www.dspace.com/en/inc/home/products/sw/system_architecture_software/systemdesk.cfm



2.4.74 SyVoLT (Symbolic Verifier of mOdeL Transformations)

SyVOLT is a contract verification tool. It is distributed as a plugin for the Eclipse development environment and allows the verification of structural pre-/post-condition contracts on model transformations.

Supported Languages

None

References

- Selim, G. M. K., L. Lúcio, J. R. Cordy, J. Dingel and B. J. Oakes (2014), Specification and Verification of Graph-Based Model Transformation Properties, in *Graph Transformation*, Eds. H. Giese and B. König, Springer International Publishing, Cham, pp. 113–129, ISBN 978-3-319-09108-2
- Selim, G. M. K., J. R. Cordy, J. Dingel, L. Lucio and B. J. Oakes (2015), Finding and Fixing Bugs in Model Transformations with Formal Verification: An Experience Report, in *AMT@MoDELS*
- Oakes, B. J., J. Troya, L. Lúcio and M. Wimmer (2018), Full contract verification for ATL using symbolic execution, *Software & Systems Modeling*, **17**, 3, pp. 815–849, ISSN 1619-1374, doi: 10.1007/s10270-016-0548-7.
<https://doi.org/10.1007/s10270-016-0548-7>
- Oakes, B. J., J. Troya, L. Lúcio and M. Wimmer (2015), Fully verifying transformation contracts for declarative ATL, in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 256–265, doi: 10.1109/MODELS.2015.7338256
- Lucio, L., B. J. Oakes, C. Gomes, G. M. K. Selim, J. Dingel, J. R. Cordy and H. Vangheluwe (2015), SyVOLT: Full Model Transformation Verification Using Contracts, in *P&DMoDELS*
- Lúcio, L., B. Oakes and H. Vangheluwe (2014), A technique for symbolically verifying properties of graph-based model transformations, *Technical report, Technical Report SOCS-TR-2014.1, McGill U*

2.4.75 TAPALL

Supported Languages

None

References

2.4.76 Time Square

TimeSquare is an Eclipse-based tool for the Clock Constraint Specification Language (CCSL). It allows the definition of CCSL specifications and the simulation of constraints and generation of a trace model.

Supported Languages

Time Square is a tool for the following languages:

- CCSL (see section 2.3.11)

References

- (2018), TimeSquare Website, <http://timesquare.inria.fr/>



2.4.77 TINA (Time petri Net Analyzer)

TINA is a model checker for Petri nets and Time Petri nets. It supports various abstract state space constructions, while preserving properties of the concrete state space representations (e.g. reachability properties, deadlock freeness, liveness).

Supported Languages

TINA is a tool for the following languages:

- CDL (see section 2.3.12)
- FIACRE (see section 2.3.18)

References

- (2018), TINA Website, <http://projects.laas.fr/tina/>

2.4.78 TTool

TTool is a tool for the modeling, analysis, formal verification and code generation of embedded systems and Systems on Chip (SoC). It allow the edition of UML and SysML diagrams supporting several UML profiles:

- DIPLODOCUS: For the partitioning of Systems-on-Chip or embedded systems. - AVATAR: For the modeling and formal verification of real-time embedded software. - SysML-Sec: For the modeling and formal verification of real-time embedded systems with security and safety issues.

Supported Languages

TTool is a tool for the following languages:

- UML (see section 2.3.69)
- SysML (see section 2.3.66)
- TEPE (see section 2.3.68)

References

- (2018), TTool Website, <http://ttool.telecom-paristech.fr/>

2.4.79 UML Analyzer

UML Analyzer is a tool for the consistency checking of UML models. It is integrated with IBM Rational Rose.

Supported Languages

UML Analyzer is a tool for the following languages:

- UML (see section 2.3.69)

References

- Egyed, A. (2007), UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models, in *Proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society, Washington, DC, USA, ICSE '07, pp. 793–796, ISBN 0-7695-2828-7, doi: 10.1109/ICSE.2007.91.
<https://doi.org/10.1109/ICSE.2007.91>
- (2018a), UML Analyzer Website, http://sunset.usc.edu/available_tools/UML_analyzer/



2.4.80 UPPAAL

UPPAAL is a modeling, simulation and model-checking tool for timed and real-time systems. It allows the desing and verification of timed automata and similar formalisms.

Supported Languages

UPPAAL is a tool for the following languages:

- UPPAAL Language (see section 2.3.71)

References

- (2018b), UPPAAL Website, <http://www.uppaal.com/>

2.4.81 Visone

Visone is a tool for the analysis and visualisation of social networks.

Supported Languages

Visone is a tool for the following languages:

- Visone Language (see section 2.3.74)

References

- (2018), Visone Website, <https://visone.info/>

2.4.82 XPPAUT

Supported Languages

XPPAUT is a tool for the following languages:

- XPPAUT Language (see section 2.3.75)

References

- (2018), XPPAUT Website, <http://www.math.pitt.edu/~bard/xpp/xpp.html>

2.4.83 Zen-RUCM (Restricted Use Case Modeling)

Zen-RUCM is an extension of the Restricted Use Case Modeling approach. Its accompanying tool allows the analysis and verification of requirements defined in specified in restricted natural language. Requirements models can be automatically generated and verified.

Supported Languages

None

References

- (2018), Zen-RUCM Website, <http://zen-tools.com/tools/zen-rucm.html>



3 Summary and Future Work

This report presents a catalog of formalisms, modelling languages and tools that are used in the domain of cyber-physical systems development. It was compiled as part of the deliverables produced by the *ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) – Working Group 1 (WG1) on Foundations of MPM4CPS*.

The catalog comprises descriptions of over 170 formalisms, languages and tools, and is automatically generated from a digital ontology that was established using modern ontology engineering techniques. The tight network of related ontology entries has been adapted to catalog form for this report. Since it is a static snapshot of the current state of art, it requires that its contents are revisited in regular intervals to extend, update and maintain the correctness of the information.

The catalog poses a solid foundation for a future, more complete body of knowledge. To this extent, an extension of the catalog and the addition of further, more detailed properties for all entries should be performed and the catalog extended. Obviously, this vast and time-consuming task cannot be performed by a small group of researchers, but instead should be executed through continuous, collaborative effort. While the descriptions and references in this catalog already provide a significant coverage of the domain, we believe that tool creators and language developers are best suited to maintain the catalog entries for their field of expertise. Evidently, this solution also requires that these experts have the means to easily create, update and alter the ontology's entries.

There also exist more concrete research tasks to be performed on the catalog. For example, in the course of the ontology development, we discovered that there exist many more properties for each individual entry that can be used to search, filter and explore the ontology. It would be of interest to look into possibilities of how to best represent this multi-dimensional information within a catalog, while maintaining its legibility and usability.

One possibility would be to group the entries into subsections, based on these properties. For example, we discovered that many tools serve particular purposes, such as e.g. analysis or simulation. This information was gathered and entered into the ontology and is readily available to be used. At the moment, we are evaluating the benefits of this approach, as for example some tools serve multiple purposes and thus would be present in multiple subsections, thereby introducing redundancy. The members of WG1 envisage a collaboration with experts in the ontology domains to harness their knowledge in this matter.

Future tasks should also establish links to the practical usage of the listed formalisms, tools and modelling languages. For instance, the MPM4CPS's Working Group 4 has compiled a metadata study Barišić et al. (2019), reporting on which formalisms and tools were used in primary studies of CPS modelling. Based on this information it can then be possible to discover e.g. common formalisms for particular CPS application domains, or e.g. list potential alternative tools and languages that can be used for certain projects.



Bibliography

- (2018), 20-sim Website, <http://20sim.com/>.
- (2018a), AADL-Inspector Website, <http://www.ellidiss.com/products/aadl-inspector/>.
- (2018b), AADL Website, <http://www.aadl.info/>.
- (2018), ACME-Studio Website, <http://www.cs.cmu.edu/~acme/AcmeStudio/>.
- (2018), AF3 Website, <http://af3.fortiss.org/>.
- (2018), Alloy Website, <http://alloy.lcs.mit.edu/alloy/>.
- (2018), AMESim Website, <https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html>.
- (2018), Any-Logic Website, <http://www.anylogic.com/>.
- (2018), Arena Website, <https://www.arenasimulation.com/>.
- (2018), ASCENS Website, <http://www.ascens-ist.eu/>.
- (2018a), Asmeta Website, <http://asmeta.sourceforge.net/>.
- (2018b), AsmGofer Website, <https://tydo.eu/AsmGofer/>.
- (2018c), AsmL Website, <https://www.microsoft.com/en-us/research/project/asml-abstract-state-machine-language/>.
- (2018), AUTOSAR Website, <https://www.autosar.org/>.
- (2018), Capella Website, <https://polarsys.org/capella/>.
- (2018), Cellular-Automata Website, <https://plato.stanford.edu/entries/cellular-automata/>.
- (2018), COMSOL Website, <https://www.comsol.com/>.
- (2018), Crescendo Website, <http://crescendotool.org/>.
- (2018), CyPhySim Website, <http://cyphysim.org/>.
- (2018), Cytoscape Website, <https://cytoscape.org/>.
- (2018), DEECo Website, <http://d3s.mff.cuni.cz/software/deeco/>.
- (2018), Dymola Website, <https://www.3ds.com/products-services/catia/products/dymola/>.
- (2018), EAST-ADL Website, <http://www.east-adl.info>.
- (2018), Entity Relationship Website, <https://www.smartdraw.com/entity-relationship-diagram/>.
- (2018), ePNK Website, www.imm.dtu.dk/~ekki/projects/ePNK/.
- (2018), FCM Website, <http://www.ec3m.net/>.
- (2018), FIACRE Website, <http://projects.laas.fr/fiacre/>.
- (2018), Flexsim Website, <https://www.flexsim.com/flexsim>.
- (2018), FUMML Website, <https://www.omg.org/spec/FUMML/>.
- (2018), HyTech Website, <https://ptolemy.berkeley.edu/projects/embedded/research/hytech/>.
- (2018), iGraph Website, <https://igraph.org/>.
- (2018), IRM Website, <http://d3s.mff.cuni.cz/software/irm/>.
- (2018), JDEECo Website, <https://github.com/d3scomp/JDEECo/wiki>.



- (2018), Kronos Website, <http://www-verimag.imag.fr/~tripakis/openkronos.html>.
- (2018), LabVIEW Website, <http://www.ni.com/en-us/shop/labview/labview-details.html>.
- (2018), MARTE Website, <http://www.omg.org/spec/MARTE/>.
- (2018a), MASIW Website, <http://www.ispras.ru/en/technologies/masiw/>.
- (2018b), MAST Website, <https://mast.unican.es/>.
- (2018a), Mathematica-Complex-Networks Website, <http://www.wolfram.com/mathematica/>.
- (2018b), Mathematica Website, <http://www.wolfram.com/mathematica/>.
- (2018), MATSim Website, <https://matsim.org/>.
- (2018), MetaH Website, <http://hop1.info/showlanguage.prx?exp=7769>.
- (2018), MODELICA Website, <https://www.modelica.org/>.
- (2018), MS4 Website, <http://www.ms4systems.com/>.
- (2018a), NetworkWorkbench Website, <http://nwb.cns.iu.edu/>.
- (2018b), NetworkX Website, <https://networkx.github.io/>.
- (2018), NuSMV Website, <http://nusmv.fbk.eu/>.
- (2018), OBPCDL Website, <http://www.obpcdl.org/>.
- (2018), Ocarina Website, <http://www.openaadl.org/ocarina.html>.
- (2018), OCL Website, <https://www.omg.org/spec/OCL>.
- (2018), OMNetPP Website, <https://www.omnetpp.org/>.
- (2018), OSATE Website, <http://osate.org/>.
- (2018), Overture Website, <https://www.overturetool.org/>.
- (2018), Pajek Website, <http://mrvar.fdv.uni-lj.si/pajek/>.
- (2018), Palladio Website, <https://www.palladio-simulator.com/home/>.
- (2018), Papyrus Website, <https://www.eclipse.org/papyrus/>.
- (2018), PHAVer Website, http://www-verimag.imag.fr/~frehse/phaver_web/.
- (2018), PNML Website, <http://www.pnml.org/>.
- (2018), Polychrony Website, <http://www.irisa.fr/espresso/Polychrony/>.
- (2018), PRISM Website, <http://www.prismmodelchecker.org>.
- (2018), PROMISE Website, <https://sergiogarcia6.wixsite.com/promise>.
- (2018a), Property Specification Patterns Website, <http://ps-patterns.wikidot.com>.
- (2018), PsALM Website, <http://178.62.206.217/patterns/psalm/>.
- (2018), PSC Website, <http://www.di.univaq.it/psc2ba/index.html>.
- (2018b), PSPWizard Website, <http://www.ict.swin.edu.au/personal/mlumpe/PSPWizard/>.
- (2018), PTC Website, <https://www.ptc.com/>.
- (2018), Ptolemy Website, <http://ptolemy.eecs.berkeley.edu/>.
- (2018), PyDSTool Website, <http://www2.gsu.edu/~matrhc/PyDSTool.htm>.
- (2018), Rainbow Website, <http://www.cs.cmu.edu/~able/research/rainbow/>.
- (2018), ROS Website, <http://www.ros.org/>.
- (2018), RSA Website, <https://www.ibm.com/developerworks/downloads/r/architect/>.



- (2018c), SAE AADL Specification, <http://standards.sae.org/as5506b/>.
- (2018), SCADE Website, <https://www.ansys.com/products/embedded-software/ansys-scade-suite>.
- (2018), Scilab Website, <https://www.scilab.org/>.
- (2018a), SimEvents Website, <https://www.mathworks.com/products/simevents.html>.
- (2018b), Simio Website, <https://www.simio.com/products/simulation-software.php>.
- (2018), SIMUL8 Website, <https://www.simul8.com/>.
- (2018), Simulink Website, <https://ch.mathworks.com/de/products/simulink.html>.
- (2018), SMT-LIB Website, <http://smtlib.cs.uiowa.edu/>.
- (2018), Spin Website, <http://spinroot.com/spin/whatispin.html>.
- (2018), Stateflow Website, <https://www.mathworks.com/products/stateflow.html>.
- (2018), STOOD Website, <https://www.ellidiss.com/products/stood/>.
- (2018a), SysML Website, <http://sysml.org/>.
- (2018b), System-Desk Website, https://www.dspace.com/en/inc/home/products/sw/system_architecture_software/systemdesk.cfm.
- (2018c), SystemC Website, <http://accellera.org/downloads/standards/systemc>.
- (2018), TimeSquare Website, <http://timesquare.inria.fr/>.
- (2018), TINA Website, <http://projects.laas.fr/tina/>.
- (2018), TTool Website, <http://ttool.telecom-paristech.fr/>.
- (2018a), UML Analyzer Website, http://sunset.usc.edu/available_tools/UML_analyzer/.
- (2018b), UML Website, <http://www.uml.org/>.
- (2018a), UPPAAL-SMC Website, <http://people.cs.aau.dk/~kg1/SSFT2015/SMC/>.
- (2018b), UPPAAL Website, <http://www.uppaal.com/>.
- (2018), Visone Website, <https://visone.info/>.
- (2018), XPPAUT Website, <http://www.math.pitt.edu/~bard/xpp/xpp.html>.
- (2018), Zen-RUCM Website, <http://zen-tools.com/tools/zen-rucm.html>.
- Abate, A., J.-P. Katoen, J. Lygeros and M. Prandini (2010), Approximate Model Checking of Stochastic Hybrid Systems, *European Journal of Control*, **16**, 6, pp. 624 – 641, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.624-641>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706919>
- Abdulla, P. A. and A. Nylén (2001), Timed Petri Nets and BQOs, in *Applications and Theory of Petri Nets 2001*, Eds. J.-M. Colom and M. Koutny, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 53–70, ISBN 978-3-540-45740-4.
- Abeywickrama, D. B., J. Combaz, V. Horký, J. Keznikl, J. Kofroň, A. L. Lafuente, M. Loreti, A. Margheri, P. Mayer, V. Monreale, U. Montanari, C. Pincioli, P. Tůma, A. Vandin and E. Vashev (2015), *Tools for Ensemble Design and Runtime*, Springer International Publishing, Cham, pp. 429–448, ISBN 978-3-319-16310-9, doi: 10.1007/978-3-319-16310-9_13.
https://doi.org/10.1007/978-3-319-16310-9_13



- Albert, R. and A.-L. Barabási (2002), Statistical mechanics of complex networks, *Rev. Mod. Phys.*, **74**, pp. 47–97, doi: 10.1103/RevModPhys.74.47.
<https://link.aps.org/doi/10.1103/RevModPhys.74.47>
- Alur, R. and D. L. Dill (1994), A Theory of Timed Automata, *Theor. Comput. Sci.*, **126**, 2, pp. 183–235, ISSN 0304-3975, doi: 10.1016/0304-3975(94)90010-8.
[http://dx.doi.org/10.1016/0304-3975\(94\)90010-8](http://dx.doi.org/10.1016/0304-3975(94)90010-8)
- Alur, R., T. A. Henzinger and P.-H. Ho (1996), Automatic symbolic verification of embedded systems, *IEEE Transactions on Software Engineering*, **22**, 3, pp. 181–201, ISSN 0098-5589, doi: 10.1109/32.489079.
- André, C. and F. Mallet (2008), Clock Constraints in UML/MARTE CCSL, Research Report RR-6540, INRIA.
<https://hal.inria.fr/inria-00280941>
- Athanassopoulos, S., C. Kaklamanis, G. Kalfountzos and E. Papaioannou (2012), Cellular Automata for Topology Control in Wireless Sensor Networks Using Matlab, in *Proceedings of the 44th IEEE Conference on Decision and Control*, volume 164, volume 164.
- Autili, M., L. Grunske, M. Lumpe, P. Pelliccione and A. Tang (2015), Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar, *IEEE Transactions on Software Engineering*, **41**, 7, pp. 620–638, ISSN 0098-5589, doi: 10.1109/TSE.2015.2398877.
- Autili, M., P. Inverardi and P. Pelliccione (2007), Graphical scenarios for specifying temporal properties: an automated approach, *Automated Software Engineering*, **14**, 3, pp. 293–340, ISSN 1573-7535, doi: 10.1007/s10515-007-0012-6.
<https://doi.org/10.1007/s10515-007-0012-6>
- Banerjee, A. and B. Sur (2014), *Electrical Linear Networks in Practice and Theory*, Springer International Publishing, Cham, pp. 411–426, ISBN 978-3-319-01147-9, doi: 10.1007/978-3-319-01147-9_15.
https://doi.org/10.1007/978-3-319-01147-9_15
- Barabási, A.-L. and R. Albert (1999), Emergence of Scaling in Random Networks, *Science*, **286**, 5439, pp. 509–512, ISSN 0036-8075, doi: 10.1126/science.286.5439.509.
<http://science.sciencemag.org/content/286/5439/509>
- Barišić, A., D. Savić, R. Al Ali, I. Ruchkin, D. Blouin, A. Cicchetti, R. Eslampanah, O. Nikiforova, M. Abshir, M. Challenger, C. Gomes, B. Tekinerdogan, F. Erata, V. Amaral and M. Gouão (2019), Systematic Literature Review on Multi-Paradigm Modelling for Cyber-Physical Systems, Technical Report WG4.4, COST Action IC1404 MPM4CPS, doi: 10.5281/zenodo.2528953.
- Barrat, A., M. Barthelemy and A. Vespignani (2008), *Dynamical Processes on Complex Networks*, Cambridge University Press, doi: 10.1017/CBO9780511791383.
- Barros, F. (2015), A Modular Representation of Fluid Stochastic Petri Nets, in *Symposium on Theory of Modeling and Simulation*.
- Barros, F. J. (2003), Dynamic Structure Multiparadigm Modeling and Simulation, *ACM Trans. Model. Comput. Simul.*, **13**, 3, pp. 259–275, ISSN 1049-3301, doi: 10.1145/937332.937335.
<http://doi.acm.org/10.1145/937332.937335>
- Barros, F. J. (2008), Semantics of Dynamic Structure Event-based Systems, in *Proceedings of the Second International Conference on Distributed Event-based Systems*, ACM, New York, NY, USA, DEBS '08, pp. 245–252, ISBN 978-1-60558-090-6, doi: 10.1145/1385989.1386020.
<http://doi.acm.org/10.1145/1385989.1386020>
- Barros, F. J. (2018), Modular representation of asynchronous geometric integrators with support for dynamic topology, *SIMULATION*, **94**, 3, pp. 259–274.



- Behrmann, G., K. G. Larsen and J. I. Rasmussen (2005), Priced Timed Automata: Algorithms and Applications, in *Formal Methods for Components and Objects*, Eds. F. S. de Boer, M. M. Bonsangue, S. Graf and W.-P. de Roever, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 162–182, ISBN 978-3-540-31939-9.
- Bengtsson, J. and W. Yi (2004), *Timed Automata: Semantics, Algorithms and Tools*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 87–124, ISBN 978-3-540-27755-2, doi: 10.1007/978-3-540-27755-2_3.
https://doi.org/10.1007/978-3-540-27755-2_3
- Benveniste, A., T. Bourke, B. Caillaud and M. Pouzet (2011), Divide and recycle: types and compilation for a hybrid synchronous language, in *Proceedings of the ACM SIGPLAN/SIGBED 2011 conference on Languages, compilers, and tools for embedded systems, LCTES 2011*, Chicago, IL, United States, doi: 10.1145/1967677.1967687.
<https://hal.inria.fr/hal-00654112>
- Bérard, B., M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen and P. Mckenzie (2001), *KRONOS — Model Checking of Real-time Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 161–168, ISBN 978-3-662-04558-9, doi: 10.1007/978-3-662-04558-9_16.
https://doi.org/10.1007/978-3-662-04558-9_16
- Berthomieu, B. and M. Diaz (1991), Modeling and verification of time dependent systems using time Petri nets, *IEEE Transactions on Software Engineering*, **17**, 3, pp. 259–273, ISSN 0098-5589, doi: 10.1109/32.75415.
- Bertrand, N., P. Bouyer, T. Brihaye, Q. Menet, C. Baier, M. Groesser and M. Jurdzinski (2014), Stochastic timed automata, *Logical Methods in Computer Science*, **10**, 4, p. 73, doi: 10.2168/LMCS-10(4:6)2014.
<https://hal.inria.fr/hal-01102368>
- Bjørner, D. and C. B. Jones (Eds.) (1978), *The Vienna Development Method: The Meta-Language*, Springer-Verlag, Berlin, Heidelberg, ISBN 3-540-08766-4.
- Black, D. C., J. Donovan, B. Bunton and A. Keist (2010), *SystemC: From the Ground Up*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN 0-387-29240-3.
- Boccaletti, S. and V. Latora (2006), The Structure and Dynamics of Complex Networks.
- Borger, E. (2005), Abstract state machines and high-level system design and analysis, *Theoretical Computer Science*, **336**, 2, pp. 205 – 207, ISSN 0304-3975.
- Börger, E. and W. Schulte (1999), *A Programmer Friendly Modular Definition of the Semantics of Java*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 353–404, ISBN 978-3-540-48737-1, doi: 10.1007/3-540-48737-9_10.
https://doi.org/10.1007/3-540-48737-9_10
- Börger, E. and R. Stärk (2003), Springer Berlin Heidelberg, Berlin, Heidelberg.
<https://doi.org/10.1007/978-3-642-18216-7>
- Bourke, T. and M. Pouzet (2013), Zélus: A Synchronous Language with ODEs, in *16th International Conference on Hybrid Systems: Computation and Control*.
- Bozga, M., C. Daws, O. Maler, A. Olivero, S. Tripakis and S. Yovine (1998), Kronos: A model-checking tool for real-time systems, in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Eds. A. P. Ravn and H. Rischel, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 298–302, ISBN 978-3-540-49792-9.
- Bozzano, M., A. Cimatti, M. Gario and A. Micheli (2015), SMT-based Validation of Timed Failure Propagation Graphs, in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI Press, AAAI'15, pp. 3724–3730, ISBN 0-262-51129-0.
<http://dl.acm.org/citation.cfm?id=2888116.2888233>



- Broenink, J. F. (1999), Introduction to Physical Systems Modelling with Bond Graphs, in *in the SiE whitebook on Simulation Methodologies*, p. 31.
- Broman, D., E. A. Lee, S. Tripakis and M. Törngren (2012), Viewpoints, Formalisms, Languages, and Tools for Cyber-physical Systems, in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, ACM, New York, NY, USA, MPM '12, pp. 49–54, ISBN 978-1-4503-1805-1, doi: 10.1145/2508443.2508452.
<http://doi.acm.org/10.1145/2508443.2508452>
- Bures, T., I. Gerostathopoulos, P. Hnetyuka, J. Keznik, M. Kit and F. Plasil (2013), DEECO: An Ensemble-based Component System, in *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, ACM, New York, NY, USA, CBSE '13, pp. 81–90, ISBN 978-1-4503-2122-8, doi: 10.1145/2465449.2465462.
<http://doi.acm.org/10.1145/2465449.2465462>
- Buxton, J. and J. Laski (1962), Control and Simulation Language, *Computer Journal*, , 5, pp. 194–199.
- Cassez, F. and K. Larsen (2000), The Impressive Power of Stopwatches, in *International Conference on Concurrency Theory*, Springer, pp. 138–152.
- Chachkov, S. and D. Buchs (2001), From formal specifications to ready-to-use software components: the concurrent object oriented Petri net approach, in *Proceedings Second International Conference on Application of Concurrency to System Design*, pp. 99–110, doi: 10.1109/CSD.2001.981768.
- Chen, P. P.-S. (2001), *The Entity Relationship Model — Toward a Unified View of Data*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 205–234, ISBN 978-3-642-48354-7, doi: 10.1007/978-3-642-48354-7_7.
https://doi.org/10.1007/978-3-642-48354-7_7
- Cheng, S.-W. and D. Garlan (2012), Stitch: A language for architecture-based self-adaptation, *Journal of Systems and Software*, **85**, 12, pp. 2860 – 2875, ISSN 0164-1212, doi: <https://doi.org/10.1016/j.jss.2012.02.060>, self-Adaptive Systems.
<http://www.sciencedirect.com/science/article/pii/S0164121212000714>
- Chopard, B. (2012), *Cellular Automata Modeling of Physical Systems*, Springer New York, New York, NY, pp. 407–433, ISBN 978-1-4614-1800-9, doi: 10.1007/978-1-4614-1800-9_27.
https://doi.org/10.1007/978-1-4614-1800-9_27
- Dahl, O.-J. and K. Nygaard (1966), SIMULA: An ALGOL-based Simulation Language, *Commun. ACM*, **9**, 9, pp. 671–678, ISSN 0001-0782, doi: 10.1145/365813.365819.
<http://doi.acm.org/10.1145/365813.365819>
- David, A., K. G. Larsen, A. Legay, U. Nyman and A. Wasowski (2010), Timed I/O Automata: A Complete Specification Theory for Real-time Systems, in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ACM, New York, NY, USA, HSCC '10, pp. 91–100, ISBN 978-1-60558-955-8, doi: 10.1145/1755952.1755967.
<http://doi.acm.org/10.1145/1755952.1755967>
- De Nicola, R., D. Latella, A. L. Lafuente, M. Loreti, A. Margheri, M. Massink, A. Morichetta, R. Pugliese, F. Tiezzi and A. Vandin (2015), *The SCEL Language: Design, Implementation, Verification*, Springer International Publishing, Cham, pp. 3–71, ISBN 978-3-319-16310-9, doi: 10.1007/978-3-319-16310-9_1.
https://doi.org/10.1007/978-3-319-16310-9_1
- Denckla, B. and P. J. Mosterman (2005), Formalizing Causal Block Diagrams for Modeling a Class of Hybrid Dynamic Systems, in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4193–4198, ISSN 0191-2216, doi: 10.1109/CDC.2005.1582820.



- Dhaussy, P., J. Auvray, S. de Belloy, F. Boniol and E. Landel (2008), Using context descriptions and property definition patterns for software formal verification, in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pp. 89–96, doi: 10.1109/ICSTW.2008.52.
- Egyed, A. (2007), UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models, in *Proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society, Washington, DC, USA, ICSE '07, pp. 793–796, ISBN 0-7695-2828-7, doi: 10.1109/ICSE.2007.91.
<https://doi.org/10.1109/ICSE.2007.91>
- Filippov, A. F. (1990), Differential Equations with Discontinuous Righthand Sides, *SIAM Review*, **32**, 2, pp. 312–315, doi: 10.1137/1032060.
<https://doi.org/10.1137/1032060>
- Fokkink, W. (2000), *Introduction to Process Algebra*, Springer-Verlag, Berlin, Heidelberg, 1st edition, ISBN 354066579X.
- Freedman, D. (1983), *Markov chains*, Springer-Verlag, New York-Berlin, ISBN 0-387-90808-0, corrected reprint of the 1971 original.
- Garlan, D., R. Monroe and D. Wile (1997), Acme: An architecture description interchange language, in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, p. 7.
- Garlan, D., R. T. Monroe and D. Wile (2000), Acme: Architectural description of component-based systems, in *Foundations of component-based systems*, Eds. G. T. Leavens and M. Sitaraman, pp. 47–68.
- Ghezzi, C., D. Mandrioli, S. Morasca and M. Pezze (1991), A unified high-level Petri net formalism for time-critical systems, *IEEE Transactions on Software Engineering*, **17**, 2, pp. 160–172, ISSN 0098-5589, doi: 10.1109/32.67597.
- Giese, H. and D. Blouin (2016), Framework to Relate / Combine Modeling Languages and Techniques, Technical Report D1.2 (Version 1), ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS).
- van Glabbeek, R. J. (1987), Bounded nondeterminism and the approximation induction principle in process algebra, in *STACS 87*, Eds. F. J. Brandenburg, G. Vidal-Naquet and M. Wirsing, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 336–347, ISBN 978-3-540-47419-7.
- Gomes, C., J. Denil and H. Vangheluwe (2016), Causal-Block Diagrams, Technical report.
<http://msdl.cs.mcgill.ca/people/claudio/pub/Gomes2016a.pdf>
- Grosu, R., M. Broy, B. Selic and G. Stefanescu (1999), *What is Behind UML-RT?*, Springer US, Boston, MA, pp. 75–90, ISBN 978-1-4615-5229-1, doi: 10.1007/978-1-4615-5229-1_6.
https://doi.org/10.1007/978-1-4615-5229-1_6
- H. Strogatz, S. (2001), Exploring Complex Networks, *Nature*, **410**, doi: 10.1038/410268a0.
<http://www.nature.com/nature/journal/v410/n6825/pdf/410268a0.pdf>
- Henzinger, T. A., P.-H. Ho and H. Wong-Toi (1997), HyTech: A model checker for hybrid systems, in *Computer Aided Verification*, Ed. O. Grumberg, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 460–463, ISBN 978-3-540-69195-2.
- Herber, P. and S. Glesner (2015), Verification of Embedded Real-time Systems, in *Formal Modeling and Verification of Cyber-Physical Systems, 1st International Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015*, pp. 1–25, doi: 10.1007/978-3-658-09994-7_1.
https://doi.org/10.1007/978-3-658-09994-7_1



- Horni, A., K. Nagel and K. Axhausen (Eds.) (2016), *Multi-Agent Transport Simulation MATSim*, Ubiquity Press, London, ISBN 978-1-909188-75-4, 978-1-909188-76-1, 978-1-909188-77-8, 978-1-909188-78-5, doi: 10.5334/baw.
- Huth, M. and M. Dermot Ryan (2000), *Logic in computer science - modelling and reasoning about systems.*, ISBN 978-0-521-65602-3.
- Huth, M. and M. Ryan (2004), *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, 2 edition, doi: 10.1017/CBO9780511810275.
- I. Utkin, V. (1992), *Sliding Modes in Control Optimization*, pp. 12–28, ISBN 978-3-642-84381-5, doi: 10.1007/978-3-642-84379-2_2.
- Jan, M., C. Jouvray, F. Kordon, A. Kung, J. Lalande, F. Loiret, J. Navas, L. Pautet, J. Pulou, A. Radermacher and L. Seinturier (2012), Flex-eWare: a Flexible MDE-based Solution for Designing and Implementing Embedded Distributed Systems, *Software: Practice and Experience*, **42**, 12, pp. 1467–1494, doi: 10.1002/spe.1143.
<https://hal.inria.fr/inria-00628310>
- Kaynar, D. K., N. Lynch, R. Segala and F. Vaandrager (2010), The theory of timed I/O automata, *Synthesis Lectures on Distributed Computing Theory*, **1**, 1, pp. 1–137.
- Keznikl, J., T. Bures, F. Plasil and M. Kit (2012), Towards Dependable Emergent Ensembles of Components: The DEECo Component Model, in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pp. 249–252, doi: 10.1109/WICSA-ECSA.212.39.
- Kiviat, P. J. (1968), Introduction to the SIMSCRIPT II Programming Language, in *Proceedings of the Second Conference on Applications of Simulations*, Winter Simulation Conference, pp. 32–36.
<http://dl.acm.org/citation.cfm?id=800166.805259>
- Kleene, S. C. (2002), *Mathematical Logic*, Wiley.
- Knorreck, D., L. Apvrille and P. D. Saqui-Sannes (2010), TEPE: A SysML language for timed-constrained property modeling and formal verification, in *Proceedings of the UML&Formal Methods Workshop (UML&FM), November 2010*, pp. 17–47.
- Liu, Z., J. Liu, J. He and Z. Ding (2012), Spatio-temporal UML Statechart for Cyber-Physical Systems, in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pp. 137–146.
- Lúcio, L., B. Oakes and H. Vangheluwe (2014), A technique for symbolically verifying properties of graph-based model transformations, *Technical report, Technical Report SOCS-TR-2014.1, McGill U.*
- Lucio, L., B. J. Oakes, C. Gomes, G. M. K. Selim, J. Dingel, J. R. Cordy and H. Vangheluwe (2015), SyVOLT: Full Model Transformation Verification Using Contracts, in *P&DMoDELS*.
- Lygeros, J. and M. Prandini (2010), Stochastic Hybrid Systems: A Powerful Framework for Complex, Large Scale Applications, *European Journal of Control*, **16**, 6, pp. 583 – 594, ISSN 0947-3580, doi: <https://doi.org/10.3166/ejc.16.583-594>.
<http://www.sciencedirect.com/science/article/pii/S0947358010706889>
- Lynch, N., R. Segala and F. Vaandrager (2003), Hybrid I/O automata, *Information and Computation*, **185**, 1, pp. 105 – 157, ISSN 0890-5401, doi: [https://doi.org/10.1016/S0890-5401\(03\)00067-1](https://doi.org/10.1016/S0890-5401(03)00067-1).
<http://www.sciencedirect.com/science/article/pii/S0890540103000671>
- Lynch, N. A. and M. R. Tuttle (1989), An introduction to input/output automata, *CWI Quarterly*, **2**, pp. 219–246.



- Misra, A. (1999), Robust diagnostic system : structural redundancy approach.
- Murthy, P. K. and E. A. Lee (2002), Multidimensional synchronous dataflow, *IEEE Transactions on Signal Processing*, **50**, 8, pp. 2064–2079, ISSN 1053-587X, doi: 10.1109/TSP.2002.800830.
- Nagel, L. W. and D. Pederson (1973), SPICE (Simulation Program with Integrated Circuit Emphasis), Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley.
<http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html>
- Navarro-Lopez, E. (2013), *DYVERSE: From formal verification to biologically-inspired real-time self-organizing systems*, pp. pp. 301–346, ISBN ISBN-10:1439883270.
- Navarro-Lopez, E. M. (2009a), Hybrid-automaton models for simulating systems with sliding motion: still a challenge, *IFAC Proceedings Volumes*, **42**, 17, pp. 322 – 327, ISSN 1474-6670, doi: <https://doi.org/10.3182/20090916-3-ES-3003.00056>, 3rd IFAC Conference on Analysis and Design of Hybrid Systems.
<http://www.sciencedirect.com/science/article/pii/S1474667015307825>
- Navarro-Lopez, E. M. (2009b), Hybrid modelling of a discontinuous dynamical system including switching control, *IFAC Proceedings Volumes*, **42**, 7, pp. 87 – 92, ISSN 1474-6670, doi: <https://doi.org/10.3182/20090622-3-UK-3004.00019>, 2nd IFAC Conference on Analysis and Control of Chaotic Systems.
<http://www.sciencedirect.com/science/article/pii/S1474667015367458>
- Neumann, R. (2014), Using Promela in a Fully Verified Executable LTL Model Checker, in *Verified Software: Theories, Tools and Experiments*, Eds. D. Giannakopoulou and D. Kroening, Springer International Publishing, Cham, pp. 105–114, ISBN 978-3-319-12154-3.
- Newman, M. (2003), The Structure and Function of Complex Networks, *SIAM Review*, **45**, 2, pp. 167–256, doi: 10.1137/S003614450342480.
- Nicola, R. D. (2013), A gentle introduction to Process Algebras.
- Niedermayer, D. (2008), *An Introduction to Bayesian Networks and Their Contemporary Applications*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 117–130, ISBN 978-3-540-85066-3, doi: 10.1007/978-3-540-85066-3_5.
https://doi.org/10.1007/978-3-540-85066-3_5
- Oakes, B. J., J. Troya, L. Lúcio and M. Wimmer (2015), Fully verifying transformation contracts for declarative ATL, in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 256–265, doi: 10.1109/MODELS.2015.7338256.
- Oakes, B. J., J. Troya, L. Lúcio and M. Wimmer (2018), Full contract verification for ATL using symbolic execution, *Software & Systems Modeling*, **17**, 3, pp. 815–849, ISSN 1619-1374, doi: 10.1007/s10270-016-0548-7.
<https://doi.org/10.1007/s10270-016-0548-7>
- Ober, I. and I. Dragomir (2010), OMEGA2: A New Version of the Profile and the Tools, in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 373–378, doi: 10.1109/ICECCS.2010.41.
- Ofsthun, S. C. and S. Abdelwahed (2007), Practical applications of timed failure propagation graphs for vehicle diagnosis, in *2007 IEEE Autotestcon*, pp. 250–259, ISSN 1088-7725, doi: 10.1109/AUTEST.2007.4374226.
- Osada, Y., T. French, M. Reynolds and H. Smallbone (2014), Hourglass Automata, *Electronic Proceedings in Theoretical Computer Science*, **161**, pp. 175–188, ISSN 2075-2180, doi:



- 10.4204/EPTCS.161.16.
- Paynter, H. M. (1961), *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, MA.
- Pnueli, A. (1977), The Temporal Logic of Programs, in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Washington, DC, USA, SFCS '77, pp. 46–57, doi: 10.1109/SFCS.1977.32.
<https://doi.org/10.1109/SFCS.1977.32>
- Porter, J., G. Hemingway, N. Kottenstette, G. Karsai and J. Sztipanovits (2011), The ESMoL Language and Tools for High-Confidence Distributed Control Systems Design . Part 1 : Design Language , Modeling Framework , and Analysis.
- Ruf, J., D. W. Hoffmann, J. Gerlach, T. Kropf, W. Rosenstiel and W. Müller (2001), The simulation semantics of systemC, in *DATE*.
- Ruijters, E. and M. Stoelinga (2015), Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools, *Computer Science Review*, **15-16**, pp. 29 – 62, ISSN 1574-0137, doi: <https://doi.org/10.1016/j.cosrev.2015.03.001>.
<http://www.sciencedirect.com/science/article/pii/S1574013715000027>
- Salem, A. (2003), Formal Semantics of Synchronous SystemC, in *Automation and Test in Europe Conference and Exhibition 2003 Design*, pp. 376–381, doi: 10.1109/DATE.2003.1253637.
- Seceleanu, C., A. Vulgarakis and P. Pettersson (2009), REMES: A resource model for embedded systems, pp. 84–94, doi: 10.1109/ICECCS.2009.49.
- Seiger, R., C. Keller, F. Niebling and T. Schlegel (2015), Modelling complex and flexible processes for smart cyber-physical environments, *Journal of Computational Science*, **10**, pp. 137 – 148, ISSN 1877-7503, doi: <https://doi.org/10.1016/j.jocs.2014.07.001>.
<http://www.sciencedirect.com/science/article/pii/S1877750314000970>
- Selim, G. M. K., J. R. Cordy, J. Dingel, L. Lucio and B. J. Oakes (2015), Finding and Fixing Bugs in Model Transformations with Formal Verification: An Experience Report, in *AMT@MoDELS*.
- Selim, G. M. K., L. Lúcio, J. R. Cordy, J. Dingel and B. J. Oakes (2014), Specification and Verification of Graph-Based Model Transformation Properties, in *Graph Transformation*, Eds. H. Giese and B. König, Springer International Publishing, Cham, pp. 113–129, ISBN 978-3-319-09108-2.
- Sigman, K. (2009), 1 IEOR 6711 : Continuous-Time Markov Chains.
- Wang, X. F. and G. Chen (2003), Complex networks: small-world, scale-free and beyond, *IEEE Circuits and Systems Magazine*, **3**, 1, pp. 6–20, ISSN 1531-636X, doi: 10.1109/MCAS.2003.1228503.
- Watts, D. and S. H. Strogatz (1998), Collective Dynamics of Small World Networks, *Nature*, **393**, pp. 440–2, doi: 10.1038/30918.
- Wile, D. (1996), Semantics for the Architecture Interchange Language, ACME, in *Joint Proceedings of the Second International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, ACM, New York, NY, USA, ISAW '96, pp. 28–30, ISBN 0-89791-867-3, doi: 10.1145/243327.243341.
<http://doi.acm.org/10.1145/243327.243341>
- Winkel, B. (2016), Nagy, G. 2015. Ordinary Differential Equations.
<https://www.simiode.org/resources/2681>
- Zeigler, B. (1976), *Theory of Modelling and Simulation*, Wiley.



Zeigler, B. (1984), *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, San Diego.

Zhang, T., K. Chakrabarty and R. Fair (2002), *Microelectrofluidic Systems: Modeling and Simulation*, Nano- and Microscience, Engineering, Technology and Medicine, CRC Press, ISBN 978-1-4200-4049-4.

Index

- 20-sim, 49
- 20-sim Block Diagrams, 22
- 20-sim Bond Graph, 22

- AADL, 22
- AADL Inspector, 49
- Abstract State Machines, 5
- ACME, 23
- ACME Studio, 50
- AF3, 50
- AF3 Language, 24
- Alloy, 24
- Alloy Analyzer, 50
- Alpina, 50
- AMESim, 50
- Any Logic, 51
- ARENA, 51
- ARENA Language, 24
- Asmeta Tools, 51
- AsmetaL, 25
- AsmGofer, 51
- AsmL, 25
- AUTOSAR Language, 25

- Bayesian Networks, 15
- Bond Graphs, 15

- Capella, 52
- Causal Block Diagrams, 15
- CCSL, 26
- CDL, 26
- Cellular Automata, 5
- Complex Networks, 16
- COMSOL Multiphysics, 52
- Crescendo, 52
- CTL, 11
- CyPhySim, 52
- Cytoscape, 52
- Cytoscape Language, 26

- Data Flow, 10
- DEECo, 27
- Differential Equations, 17
- Discontinuous Systems, 17
- Discrete Event, 18
- Dymola, 53

- EAST-ADL, 27

- Electrical Linear Networks, 19
- Entity Relationship, 19
- Entity Relationship Diagram, 27
- ePNK, 53
- ESMoL, 53

- Fault Trees, 20
- FCM, 28
- FIACRE, 28
- First Order Logic, 12
- FlexSim, 53
- FlexSim Language, 29
- FlyAQ, 54
- Formalisms
 - Abstract State Machines, 5
 - Bayesian Networks, 15
 - Bond Graphs, 15
 - Causal Block Diagrams, 15
 - Cellular Automata, 5
 - Complex Networks, 16
 - CTL, 11
 - Data Flow, 10
 - Differential Equations, 17
 - Discontinuous Systems, 17
 - Discrete Event, 18
 - Electrical Linear Networks, 19
 - Entity Relationship, 19
 - Fault Trees, 20
 - First Order Logic, 12
 - High Level Petri Nets, 13
 - Hybrid Automata, 6
 - HyFlow, 11
 - I/O Hybrid Automata, 6
 - Linear Hybrid Automata, 7
 - Linear Temporal Logic, 12
 - Markov Chains, 20
 - Non-Linear Hybrid Automata, 7
 - Petri Net, 13
 - Petri Nets with Priority, 14
 - Process Algebras, 20
 - PTAs, 8
 - Stochastic Hybrid Automata, 8
 - Stochastic Petri Nets, 14
 - Stochastic Timed Automata, 9
 - Temporal Logic, 12
 - TFPG, 21
 - Timed Automata, 9



- Timed I/O Automata, 10
- Timed Petri Nets, 14
- VDM, 21
- FUML, 29
- Gofer, 29
- GreatSPN, 54
- High Level Petri Nets, 13
- Hybrid Automata, 6
- HyFlow, 11
- HyTech, 54
- HyTech Language, 29
- I/O Hybrid Automata, 6
- iGraph, 54
- iGraph Language, 30
- IRM, 30
- IRM SA, 55
- IRM SA Language, 30
- jDEECo, 55
- Kronos, 55
- Kronos Language, 31
- LabVIEW, 56
- LabVIEW Language, 31
- Languages
 - 20-sim Block Diagrams, 22
 - 20-sim Bond Graph, 22
 - AADL, 22
 - ACME, 23
 - AF3 Language, 24
 - Alloy, 24
 - ARENA Language, 24
 - AsmetaL, 25
 - AsmL, 25
 - AUTOSAR Language, 25
 - CCSL, 26
 - CDL, 26
 - Cytoscape Language, 26
 - DEECo, 27
 - EAST-ADL, 27
 - Entity Relationship Diagram, 27
 - FCM, 28
 - FIACRE, 28
 - FlexSim Language, 29
 - FUML, 29
 - Gofer, 29
 - HyTech Language, 29
 - iGraph Language, 30
 - IRM, 30
 - IRM SA Language, 30
 - Kronos Language, 31
 - LabVIEW Language, 31
 - Lustre, 31
 - MARTE, 32
 - Mathematica Language, 32
 - MetaH, 32
 - Modelica, 33
 - MS4 Me Language, 33
 - Network Workbench Language, 34
 - NetworkX Language, 34
 - NuSMV Language, 34
 - OCL, 35
 - OMEGA2, 35
 - OMNet++ Language, 35
 - Pajek Language, 35
 - Palladio Component Model, 36
 - Petri Net Diagram, 36
 - PNML, 37
 - PRISM Language, 37
 - Promela, 37
 - PROMISE Language, 38
 - PSC, 38
 - PSPF Language, 39
 - Ptolemy Language, 39
 - PyDSTool Language, 39
 - Random-Graph Networks, 40
 - Scale-Free Networks, 40
 - SCEL, 40
 - Scilab Language, 41
 - Signal, 41
 - SimEvents Language, 42
 - Simio Language, 42
 - SIMUL8 Language, 42
 - Simulink Language, 42
 - Small-World Networks, 43
 - SMT-LIB, 43
 - Spatio-Temporal UML Statechart, 43
 - SPICE Language, 44
 - Stateflow Language, 44
 - Stitch, 44
 - SysML, 45
 - SystemC, 45
 - TEPE, 46
 - UML, 46
 - UML-RT, 47
 - UPPAAL Language, 47
 - UPPAAL SMC Language, 47
 - VDM-SL, 48
 - Visone Language, 48
 - XPPAUT Language, 48



- Zélus, 48
Linear Hybrid Automata, 7
Linear Temporal Logic, 12
Lola, 56
Lustre, 31
- Markov Chains, 20
MARTE, 32
MASIW, 56
MAST, 56
MAT Sim, 56
Mathematica, 57
Mathematica Language, 32
MetaH, 32
Modelica, 33
MS4 Me, 57
MS4 Me Language, 33
- Network Workbench, 57
Network Workbench Language, 34
NetworkX, 57
NetworkX Language, 34
Non-Linear Hybrid Automata, 7
NuSMV, 58
NuSMV Language, 34
- OBP Explorer, 58
Ocarina, 58
OCL, 35
OMEGA2, 35
OMNeT++, 58
OMNet++ Language, 35
Open Modelica, 58
ORIS, 59
OSATE, 59
Overture, 59
- Pajek, 59
Pajek Language, 35
Palladio, 60
Palladio Component Model, 36
Papyrus, 60
Petri Net, 13
Petri Net Diagram, 36
Petri Nets with Priority, 14
PHAVer, 60
PNML, 37
Polychrony, 60
PragmaDevTracer, 61
PRISM, 61
PRISM Language, 37
Process Algebras, 20
- Promela, 37
PROMISE, 61
PROMISE Language, 38
PsALM, 61
PSC, 38
PSPF Language, 39
PSPWizard, 62
PTAs, 8
PTC MBSE, 62
Ptolemy, 62
Ptolemy Language, 39
PyDSTool, 63
PyDSTool Language, 39
- Rainbow, 63
Random-Graph Networks, 40
Rational Software Architect, 63
Remes, 63
ROS, 64
- Scade Suite, 64
Scale-Free Networks, 40
SCEL, 40
Scilab, 64
Scilab Language, 41
Signal, 41
SimEvents, 64
SimEvents Language, 42
Simio, 64
Simio Language, 42
SIMUL8, 65
SIMUL8 Language, 42
Simulink, 65
Simulink Language, 42
Small-World Networks, 43
SMT-LIB, 43
Spatio-Temporal UML Statechart, 43
SPICE, 65
SPICE Language, 44
Spin, 65
Stateflow, 66
Stateflow Language, 44
Stitch, 44
Stochastic Hybrid Automata, 8
Stochastic Petri Nets, 14
Stochastic Timed Automata, 9
STOOD, 66
StrataGEM, 66
SysML, 45
System Desk, 66
SystemC, 45



- SyVoLT, 67
- TAPALL, 67
- Temporal Logic, 12
- TEPE, 46
- TFPG, 21
- Time Square, 67
- Timed Automata, 9
- Timed I/O Automata, 10
- Timed Petri Nets, 14
- TINA, 68
- Tools
 - 20-sim, 49
 - AADL Inspector, 49
 - ACME Studio, 50
 - AF3, 50
 - Alloy Analyzer, 50
 - Alpina, 50
 - AMESim, 50
 - Any Logic, 51
 - ARENA, 51
 - Asmeta Tools, 51
 - AsmGofer, 51
 - Capella, 52
 - COMSOL Multiphysics, 52
 - Crescendo, 52
 - CyPhySim, 52
 - Cytoscape, 52
 - Dymola, 53
 - ePNK, 53
 - ESMoL, 53
 - FlexSim, 53
 - FlyAQ, 54
 - GreatSPN, 54
 - HyTech, 54
 - iGraph, 54
 - IRM SA, 55
 - jDEECo, 55
 - Kronos, 55
 - LabVIEW, 56
 - Lola, 56
 - MASIW, 56
 - MAST, 56
 - MAT Sim, 56
 - Mathematica, 57
 - MS4 Me, 57
 - Network Workbench, 57
 - NetworkX, 57
 - NuSMV, 58
 - OBP Explorer, 58
 - Ocarina, 58
 - OMNeT++, 58
 - Open Modelica, 58
 - ORIS, 59
 - OSATE, 59
 - Overture, 59
 - Pajek, 59
 - Palladio, 60
 - Papyrus, 60
 - PHAVer, 60
 - Polychrony, 60
 - PragmaDevTracer, 61
 - PRISM, 61
 - PROMISE, 61
 - PsALM, 61
 - PSPWizard, 62
 - PTC MBSE, 62
 - Ptolemy, 62
 - PyDSTool, 63
 - Rainbow, 63
 - Rational Software Architect, 63
 - Remes, 63
 - ROS, 64
 - Scade Suite, 64
 - Scilab, 64
 - SimEvents, 64
 - Simio, 64
 - SIMUL8, 65
 - Simulink, 65
 - SPICE, 65
 - Spin, 65
 - Stateflow, 66
 - STOOD, 66
 - StrataGEM, 66
 - System Desk, 66
 - SyVoLT, 67
 - TAPALL, 67
 - Time Square, 67
 - TINA, 68
 - TTool, 68
 - UML Analyzer, 68
 - UPPAAL, 69
 - Visone, 69
 - XPPAUT, 69
 - Zen-RUCM, 69
- TTool, 68
- UML, 46
- UML Analyzer, 68
- UML-RT, 47
- UPPAAL, 69
- UPPAAL Language, 47



UPPAAL SMC Language, 47

VDM, 21

VDM-SL, 48

Visone, 69

Visone Language, 48

XPPAUT, 69

XPPAUT Language, 48

Zélus, 48

Zen-RUCM, 69