

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

Rima Al Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, Dominique Blouin, Holger Giese, Miguel Goulao, Mauro Iacono, Navarro Eva, Hans Vangheluwe, et al.

▶ **To cite this version:**

Rima Al Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, Dominique Blouin, et al.. State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development. [Technical Report] COST European Cooperation in Science and Technology. 2017. hal-03168840

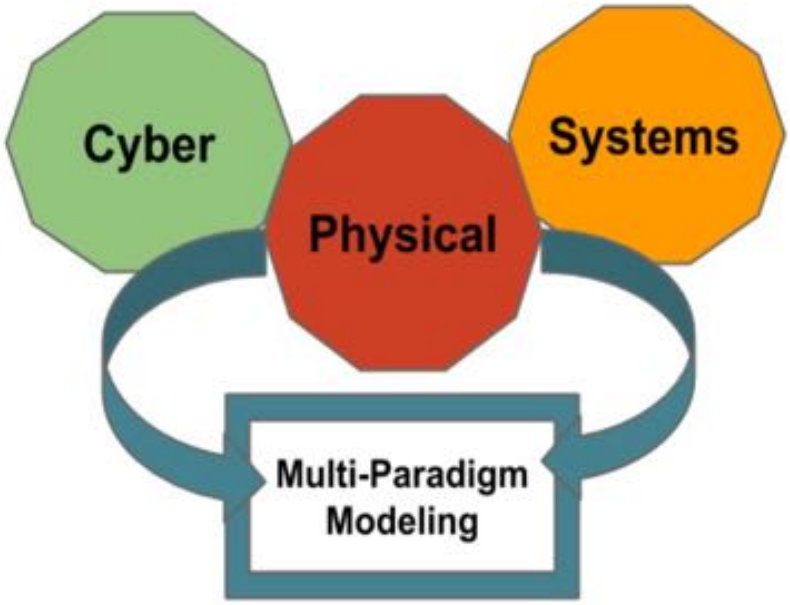
HAL Id: hal-03168840
https://hal.science/hal-03168840

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.

12012345678910



cost
EUROPEAN COOPERATION
IN SCIENCE AND TECHNOLOGY

ICT COST Action IC1404

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

Rima Al-Ali, Moussa Amrani, Ankica Barisic, Fernando Barros, Dominique Blouin, Holger Giese, Miguel Goulão, Mauro Iacono, Eva Navarro, Hans Vangheluwe, Ken Vanherpen, Manuel Wimmer

Deliverable: WG1.1

Core Team
University of Antwerp
New University of Lisbon
University of Malaga
Hasso-Plattner Inst., Potsdam
University of Twente
fortiss GmbH, München
University of Lisbon

Document Info

Deliverable	WG1.1
Dissemination	Restricted
Status	Final
Doc's Lead Partner	Hasso-Plattner Inst.
Date	January 8, 2017
Version	2.0
Pages	61



Contents

1 Introduction **1**

2 Structured Catalog of Modeling Languages and Tools **3**

2.1 Introduction . . . 3

2.2 Formalisms . . . 4

2.2.1 AbstractStateMachines . . . 4

2.2.1.1 Implementing Languages . . . 4

2.2.1.2 References . . . 4

2.2.2 BayesianNetworks . . . 4

2.2.2.1 Implementing Languages . . . 4

2.2.2.2 References . . . 4

2.2.3 CTLSpecification . . . 4

2.2.3.1 Implementing Languages . . . 4

2.2.3.2 References . . . 5

2.2.4 CausalBlockDiagrams . . . 5

2.2.4.1 Implementing Languages . . . 5

2.2.4.2 References . . . 5

2.2.5 CellularAutomata . . . 5

2.2.5.1 Implementing Languages . . . 5

2.2.5.2 References . . . 5

2.2.6 DEECo . . . 5

2.2.6.1 Implementing Languages . . . 5

2.2.6.2 References . . . 5

2.2.7 DEECoSpecification . . . 5

2.2.7.1 Implementing Languages . . . 5

2.2.7.2 References . . . 6

2.2.8 DataFlow . . . 6

2.2.8.1 Implementing Languages . . . 6

2.2.8.2 References . . . 6

2.2.9 DataFlowTimed . . . 6

2.2.9.1 Implementing Languages . . . 7

2.2.9.2 References . . . 7

2.2.10 DifferentialEquations . . . 7

2.2.10.1 Implementing Languages . . . 7

2.2.10.2 References . . . 7

2.2.11 DiscreteEvent . . . 7

2.2.11.1 Implementing Languages . . . 7
2.2.11.2 References . . . 7
2.2.12 ElectricalLinearNetworks . . . 7
2.2.12.1 Implementing Languages . . . 7
2.2.12.2 References . . . 7
2.2.13 EntityRelationship . . . 8
2.2.13.1 Implementing Languages . . . 8
2.2.13.2 References . . . 8
2.2.14 Fault Trees . . . 8
2.2.14.1 Implementing Languages . . . 8
2.2.14.2 References . . . 8
2.2.15 FiniteStateProcess . . . 8
2.2.15.1 Implementing Languages . . . 8
2.2.15.2 References . . . 8
2.2.16 First Order Logic . . . 8
2.2.16.1 Implementing Languages . . . 8
2.2.16.2 References . . . 8
2.2.17 HyFlow (Hybrid Flow System Specification) . . . 8
2.2.17.1 Implementing Languages . . . 9
2.2.17.2 References . . . 9
2.2.18 DiscontinuousSystems . . . 9
2.2.18.1 Implementing Languages . . . 9
2.2.18.2 References . . . 9
2.2.19 HybridAutomata . . . 9
2.2.19.1 LinearHybridAutomata . . . 9
2.2.19.2 NonLinearHybridAutomata . . . 10
2.2.19.3 StochasticHybridAutomata . . . 10
2.2.19.4 TimedAutomata . . . 10
2.2.19.5 TimeAutomataPriced (Priced/Probabilistic Timed Automata (PTAs)) . . . 10
2.2.19.6 TimedAutomataStochastic . . . 10
2.2.19.7 I/O_Automata . . . 11
2.2.19.8 Implementing Languages . . . 11
2.2.19.9 References . . . 11
2.2.20 LabelledTransitionSystem . . . 11
2.2.20.1 Implementing Languages . . . 11
2.2.20.2 References . . . 11
2.2.21 LinearSignalFlow . . . 11
2.2.21.1 Implementing Languages . . . 11

2.2.21.2 References . . . 11
2.2.22 MarkovChains . . . 11
2.2.22.1 Implementing Languages . . . 12
2.2.22.2 References . . . 12
2.2.23 MessageDescriptionSpecification . . . 12
2.2.23.1 Implementing Languages . . . 12
2.2.23.2 References . . . 12
2.2.24 PetriNet . . . 12
2.2.24.1 Supported Extended Formalisms . . . 12
2.2.24.2 Supporting Tools . . . 13
2.2.24.3 References . . . 13
2.2.24.4 Implementing Languages . . . 13
2.2.24.5 References . . . 13
2.2.25 PetriNetColoured . . . 13
2.2.25.1 Implementing Languages . . . 13
2.2.25.2 References . . . 14
2.2.26 PetriNetDualistic . . . 14
2.2.26.1 Implementing Languages . . . 14
2.2.26.2 References . . . 14
2.2.27 PetriNetPrioritised . . . 14
2.2.27.1 Implementing Languages . . . 14
2.2.27.2 References . . . 14
2.2.28 PetriNetStochastic . . . 14
2.2.28.1 Implementing Languages . . . 14
2.2.28.2 References . . . 14
2.2.29 PetriNetTimed . . . 14
2.2.29.1 Implementing Languages . . . 14
2.2.29.2 References . . . 14
2.2.30 ProcessAlgebras . . . 14
2.2.30.1 Implementing Languages . . . 14
2.2.30.2 References . . . 15
2.2.31 TFPG (Timed Failure Propagation Graph) . . . 15
2.2.31.1 Implementing Languages . . . 15
2.2.31.2 References . . . 15
2.2.31.3 Implementing Languages . . . 15
2.2.31.4 References . . . 15
2.2.32 Complex Networks . . . 15
2.2.32.1 Implementing Languages . . . 15

2.2.32.2 References . . . 15
2.3 Languages . . . 15
2.3.1 AADL (Architecture Analysis and Design Language) . . . 15
2.3.1.1 Supported Formalisms . . . 16
2.3.1.2 Supporting Tools . . . 16
2.3.1.3 References . . . 16
2.3.2 ACME (Architecture Description Interchange Language) . . . 16
2.3.2.1 Supported Formalisms . . . 16
2.3.2.2 Supporting Tools . . . 16
2.3.2.3 References . . . 16
2.3.3 AML . . . 16
2.3.3.1 Supported Formalisms . . . 17
2.3.3.2 Supporting Tools . . . 17
2.3.3.3 References . . . 17
2.3.4 ATL . . . 17
2.3.4.1 Supported Formalisms . . . 17
2.3.4.2 Supporting Tools . . . 17
2.3.4.3 References . . . 17
2.3.5 AUTOSARLanguage (AUTomotive Open System ARchitecture) . . . 17
2.3.5.1 Supported Formalisms . . . 17
2.3.5.2 Supporting Tools . . . 17
2.3.5.3 References . . . 17
2.3.6 Alloy . . . 17
2.3.6.1 Supported Formalisms . . . 17
2.3.6.2 Supporting Tools . . . 17
2.3.6.3 References . . . 17
2.3.7 Artisan . . . 17
2.3.7.1 Supported Formalisms . . . 17
2.3.7.2 Supporting Tools . . . 18
2.3.7.3 References . . . 18
2.3.8 AsmL (Abstract State Machine Language) . . . 18
2.3.8.1 Supported Formalisms . . . 18
2.3.8.2 Supporting Tools . . . 18
2.3.8.3 References . . . 18
2.3.9 AsmetaL . . . 18
2.3.9.1 Supported Formalisms . . . 18
2.3.9.2 Supporting Tools . . . 18
2.3.9.3 References . . . 18

2.3.10 BlockDiagram . . . 18
2.3.10.1 Supported Formalisms . . . 18
2.3.10.2 Supporting Tools . . . 18
2.3.10.3 References . . . 19
2.3.11 BondGraph . . . 19
2.3.11.1 Supported Formalisms . . . 19
2.3.11.2 Supporting Tools . . . 19
2.3.11.3 References . . . 19
2.3.12 C . . . 19
2.3.12.1 Supported Formalisms . . . 19
2.3.12.2 Supporting Tools . . . 19
2.3.12.3 References . . . 19
2.3.13 C++ . . . 19
2.3.13.1 Supported Formalisms . . . 19
2.3.13.2 Supporting Tools . . . 19
2.3.13.3 References . . . 19
2.3.14 CCSL (Clock Constraint Specification Language) . . . 19
2.3.14.1 Supported Formalisms . . . 19
2.3.14.2 Supporting Tools . . . 19
2.3.14.3 References . . . 20
2.3.15 CDL (Context Description Language) . . . 20
2.3.15.1 Supported Formalisms . . . 20
2.3.15.2 Supporting Tools . . . 20
2.3.15.3 References . . . 20
2.3.16 CTL (Computation Tree Logic) . . . 20
2.3.16.1 Supported Formalisms . . . 20
2.3.16.2 Supporting Tools . . . 20
2.3.16.3 References . . . 20
2.3.17 Clafer . . . 20
2.3.17.1 Supported Formalisms . . . 20
2.3.17.2 Supporting Tools . . . 20
2.3.17.3 References . . . 21
2.3.18 CoCoME . . . 21
2.3.18.1 Supported Formalisms . . . 21
2.3.18.2 Supporting Tools . . . 21
2.3.18.3 References . . . 21
2.3.19 DEECoDSL (Dependable Emergent Ensembles of Component-Domain Specific Language) . . . 21
2.3.19.1 Supported Formalisms . . . 21

2.3.19.2 Supporting Tools . . . 21
2.3.19.3 References . . . 21
2.3.20 DSLTrans . . . 21
2.3.20.1 Supported Formalisms . . . 21
2.3.20.2 Supporting Tools . . . 21
2.3.20.3 References . . . 21
2.3.21 EAST-ADL . . . 21
2.3.21.1 Supported Formalisms . . . 22
2.3.21.2 Supporting Tools . . . 22
2.3.21.3 References . . . 22
2.3.22 ECL (Epsilon Comparison Language) . . . 22
2.3.22.1 Supported Formalisms . . . 22
2.3.22.2 Supporting Tools . . . 22
2.3.22.3 References . . . 22
2.3.23 EGL (Epsilon Generation Language) . . . 22
2.3.23.1 Supported Formalisms . . . 22
2.3.23.2 Supporting Tools . . . 22
2.3.23.3 References . . . 22
2.3.24 EML (Epsilon Merging Language) . . . 22
2.3.24.1 Supported Formalisms . . . 22
2.3.24.2 Supporting Tools . . . 22
2.3.24.3 References . . . 23
2.3.25 EOL (Epsilon Object Language) . . . 23
2.3.25.1 Supported Formalisms . . . 23
2.3.25.2 Supporting Tools . . . 23
2.3.25.3 References . . . 23
2.3.26 ERD (Entity Relationship Diagram) . . . 23
2.3.26.1 Supported Formalisms . . . 23
2.3.26.2 Supporting Tools . . . 23
2.3.26.3 References . . . 23
2.3.27 ETL (Epsilon Transformation Language) . . . 23
2.3.27.1 Supported Formalisms . . . 23
2.3.27.2 Supporting Tools . . . 23
2.3.27.3 References . . . 23
2.3.28 EVL (Epsilon Validation Language) . . . 23
2.3.28.1 Supported Formalisms . . . 23
2.3.28.2 Supporting Tools . . . 23
2.3.28.3 References . . . 24

2.3.29 EWL (Epsilon Wizard Language) . . . 24
2.3.29.1 Supported Formalisms . . . 24
2.3.29.2 Supporting Tools . . . 24
2.3.29.3 References . . . 24
2.3.30 EclipseEGL . . . 24
2.3.30.1 Supported Formalisms . . . 24
2.3.30.2 Supporting Tools . . . 24
2.3.30.3 References . . . 24
2.3.31 EpsilonFlock . . . 24
2.3.31.1 Supported Formalisms . . . 24
2.3.31.2 Supporting Tools . . . 24
2.3.31.3 References . . . 25
2.3.32 FIACRE . . . 25
2.3.32.1 Supported Formalisms . . . 25
2.3.32.2 Supporting Tools . . . 25
2.3.32.3 References . . . 25
2.3.33 FUML (Foundational Subset for Executable UML Models) . . . 25
2.3.33.1 Supported Formalisms . . . 25
2.3.33.2 Supporting Tools . . . 25
2.3.33.3 References . . . 25
2.3.34 IRM . . . 25
2.3.34.1 Supported Formalisms . . . 25
2.3.34.2 Supporting Tools . . . 26
2.3.34.3 References . . . 26
2.3.35 IRM-SA . . . 26
2.3.35.1 Supported Formalisms . . . 26
2.3.35.2 Supporting Tools . . . 26
2.3.35.3 References . . . 26
2.3.36 IconicDiagrams . . . 26
2.3.36.1 Supported Formalisms . . . 26
2.3.36.2 Supporting Tools . . . 26
2.3.36.3 References . . . 26
2.3.37 Java . . . 26
2.3.37.1 Supported Formalisms . . . 26
2.3.37.2 Supporting Tools . . . 26
2.3.37.3 References . . . 26
2.3.38 LTL (Linear Temporal Logic) . . . 26
2.3.38.1 Supported Formalisms . . . 27

2.3.38.2 Supporting Tools . . . 27
2.3.38.3 References . . . 27
2.3.39 MARTE (Modeling and Analysis of Real-Time and Embedded systems) . . . 27
2.3.39.1 Supported Formalisms . . . 27
2.3.39.2 Supporting Tools . . . 27
2.3.39.3 References . . . 27
2.3.40 MTL (Model to Text Language) . . . 27
2.3.40.1 Supported Formalisms . . . 27
2.3.40.2 Supporting Tools . . . 27
2.3.40.3 References . . . 27
2.3.41 MessagesDescriptionLanguage . . . 27
2.3.41.1 Supported Formalisms . . . 27
2.3.41.2 Supporting Tools . . . 28
2.3.41.3 References . . . 28
2.3.42 MetaH . . . 28
2.3.42.1 Supported Formalisms . . . 28
2.3.42.2 Supporting Tools . . . 28
2.3.42.3 References . . . 28
2.3.43 MoTiF . . . 28
2.3.43.1 Supported Formalisms . . . 28
2.3.43.2 Supporting Tools . . . 28
2.3.43.3 References . . . 28
2.3.44 Modelica . . . 28
2.3.44.1 Supported Formalisms . . . 28
2.3.44.2 Supporting Tools . . . 28
2.3.44.3 References . . . 29
2.3.45 ModelicaML . . . 29
2.3.45.1 Supported Formalisms . . . 29
2.3.45.2 Supporting Tools . . . 29
2.3.45.3 References . . . 29
2.3.46 NaturalLanguage . . . 29
2.3.46.1 Supported Formalisms . . . 29
2.3.46.2 Supporting Tools . . . 29
2.3.46.3 References . . . 29
2.3.47 NuSMVLanguage . . . 29
2.3.47.1 Supported Formalisms . . . 29
2.3.47.2 Supporting Tools . . . 29
2.3.47.3 References . . . 29

2.3.48 OCL . . . 29
2.3.48.1 Supported Formalisms . . . 29
2.3.48.2 Supporting Tools . . . 30
2.3.48.3 References . . . 30
2.3.49 OMEGA2 . . . 30
2.3.49.1 Supported Formalisms . . . 30
2.3.49.2 Supporting Tools . . . 30
2.3.49.3 References . . . 30
2.3.50 OSATE2 . . . 30
2.3.50.1 Supported Formalisms . . . 30
2.3.50.2 Supporting Tools . . . 30
2.3.50.3 References . . . 30
2.3.51 PRISMLanguage . . . 30
2.3.51.1 Supported Formalisms . . . 30
2.3.51.2 Supporting Tools . . . 30
2.3.51.3 References . . . 31
2.3.52 ParallelAssignmentLanguage . . . 31
2.3.52.1 Supported Formalisms . . . 31
2.3.52.2 Supporting Tools . . . 31
2.3.52.3 References . . . 31
2.3.53 PetriNetLanguage . . . 31
2.3.53.1 Supported Formalisms . . . 31
2.3.53.2 Supporting Tools . . . 31
2.3.53.3 References . . . 32
2.3.54 ProMoBox . . . 32
2.3.54.1 Supported Formalisms . . . 32
2.3.54.2 Supporting Tools . . . 32
2.3.54.3 References . . . 32
2.3.55 Promela . . . 32
2.3.55.1 Supported Formalisms . . . 32
2.3.55.2 Supporting Tools . . . 32
2.3.55.3 References . . . 32
2.3.56 PtidyOS . . . 32
2.3.56.1 Supported Formalisms . . . 32
2.3.56.2 Supporting Tools . . . 32
2.3.56.3 References . . . 32
2.3.57 QVT . . . 32
2.3.57.1 Supported Formalisms . . . 32

2.3.57.2 Supporting Tools . . . 32
2.3.57.3 References . . . 33
2.3.58 Reo_Coordination_Language . . . 33
2.3.58.1 Supported Formalisms . . . 33
2.3.58.2 Supporting Tools . . . 33
2.3.58.3 References . . . 33
2.3.59 SMT_LIB . . . 33
2.3.59.1 Supported Formalisms . . . 33
2.3.59.2 Supporting Tools . . . 33
2.3.59.3 References . . . 33
2.3.60 STUML (Spatio-Temporal UML Statechart) . . . 33
2.3.60.1 Supported Formalisms . . . 33
2.3.60.2 Supporting Tools . . . 33
2.3.60.3 References . . . 33
2.3.61 SimPL . . . 33
2.3.61.1 Supported Formalisms . . . 33
2.3.61.2 Supporting Tools . . . 33
2.3.61.3 References . . . 34
2.3.62 SimulinkLanguage . . . 34
2.3.62.1 Supported Formalisms . . . 34
2.3.62.2 Supporting Tools . . . 34
2.3.62.3 References . . . 34
2.3.63 Stitch . . . 34
2.3.63.1 Supported Formalisms . . . 34
2.3.63.2 Supporting Tools . . . 34
2.3.63.3 References . . . 34
2.3.64 SysML (Systems Modeling Language) . . . 34
2.3.64.1 Supported Formalisms . . . 34
2.3.64.2 Supporting Tools . . . 34
2.3.64.3 References . . . 34
2.3.65 SystemCSpecification . . . 34
2.3.65.1 Implementing Languages . . . 35
2.3.65.2 References . . . 35
2.3.66 SystemC . . . 35
2.3.66.1 Supported Formalisms . . . 36
2.3.66.2 Supporting Tools . . . 36
2.3.66.3 References . . . 36
2.3.67 TCTL (Timed Computation Tree Logic) . . . 36

2.3.67.1 Supported Formalisms . . . 36
2.3.67.2 Supporting Tools . . . 36
2.3.67.3 References . . . 36
2.3.68 TEPE (Temporal Property Expression Language) . . . 36
2.3.68.1 Supported Formalisms . . . 36
2.3.68.2 Supporting Tools . . . 36
2.3.68.3 References . . . 36
2.3.69 TimedTransitionSystemLanguage . . . 36
2.3.69.1 Supported Formalisms . . . 36
2.3.69.2 Supporting Tools . . . 36
2.3.69.3 References . . . 37
2.3.70 UML (Unified Modeling Language) . . . 37
2.3.70.1 Supported Formalisms . . . 37
2.3.70.2 Supporting Tools . . . 37
2.3.70.3 References . . . 37
2.3.71 UML-RT . . . 37
2.3.71.1 Supported Formalisms . . . 37
2.3.71.2 Supporting Tools . . . 37
2.3.71.3 References . . . 37
2.3.72 UMLMARTE . . . 37
2.3.72.1 Supported Formalisms . . . 37
2.3.72.2 Supporting Tools . . . 37
2.3.72.3 References . . . 37
2.3.73 UMLProfile . . . 37
2.3.73.1 Supported Formalisms . . . 37
2.3.73.2 Supporting Tools . . . 38
2.3.73.3 References . . . 38
2.3.74 UMLSysML . . . 38
2.3.74.1 Supported Formalisms . . . 38
2.3.74.2 Supporting Tools . . . 38
2.3.74.3 References . . . 38
2.3.75 UPPAALRequirementSpecificationLanguage . . . 38
2.3.75.1 Supported Formalisms . . . 38
2.3.75.2 Supporting Tools . . . 38
2.3.75.3 References . . . 38
2.3.76 UPPAALSMCSpecificationLanguage . . . 38
2.3.76.1 Supported Formalisms . . . 38
2.3.76.2 Supporting Tools . . . 38

2.3.76.3 References . . . 38
2.3.77 VDM-SL . . . 38
2.3.77.1 Supported Formalisms . . . 39
2.3.77.2 Supporting Tools . . . 39
2.3.77.3 References . . . 39
2.3.78 Xtend . . . 39
2.3.78.1 Supported Formalisms . . . 39
2.3.78.2 Supporting Tools . . . 39
2.3.78.3 References . . . 39
2.3.79 xtext . . . 39
2.3.79.1 Supported Formalisms . . . 39
2.3.79.2 Supporting Tools . . . 39
2.3.79.3 References . . . 39
2.4 Tools . . . 39
2.4.1 20Sim . . . 39
2.4.1.1 Supported Languages . . . 40
2.4.1.2 References . . . 40
2.4.2 AADLInspector . . . 40
2.4.2.1 Supported Languages . . . 40
2.4.2.2 References . . . 40
2.4.3 AF3 . . . 40
2.4.3.1 Supported Languages . . . 40
2.4.3.2 References . . . 40
2.4.4 AMESim (AMESim (Advanced Modeling Environment for Simulations)) . . . 40
2.4.4.1 Supported Languages . . . 40
2.4.4.2 References . . . 40
2.4.5 AToM3 . . . 40
2.4.5.1 Supported Languages . . . 40
2.4.5.2 References . . . 40
2.4.6 AToMPM . . . 40
2.4.6.1 Supported Languages . . . 41
2.4.6.2 References . . . 41
2.4.7 AVATAR (AVATAR stands for Automated Verification of reAl Time softwARe.) 41
2.4.7.1 Supported Languages . . . 41
2.4.7.2 References . . . 41
2.4.8 Acceleo . . . 41
2.4.8.1 Supported Languages . . . 41
2.4.8.2 References . . . 42

2.4.9 AcmeStudio . . . 42
2.4.9.1 Supported Languages . . . 42
2.4.9.2 References . . . 42
2.4.10 AlloyTool . . . 42
2.4.10.1 Supported Languages . . . 42
2.4.10.2 References . . . 42
2.4.11 AnyLogic . . . 42
2.4.11.1 Supported Languages . . . 42
2.4.11.2 References . . . 43
2.4.12 ArcGIS . . . 43
2.4.12.1 Supported Languages . . . 43
2.4.12.2 References . . . 43
2.4.13 Asmeta . . . 43
2.4.13.1 Supported Languages . . . 43
2.4.13.2 References . . . 43
2.4.14 CHESS . . . 43
2.4.14.1 Supported Languages . . . 43
2.4.14.2 References . . . 43
2.4.15 COMSOL . . . 43
2.4.15.1 Supported Languages . . . 43
2.4.15.2 References . . . 43
2.4.16 Capella . . . 43
2.4.16.1 Supported Languages . . . 43
2.4.16.2 References . . . 44
2.4.17 Crescendo . . . 44
2.4.17.1 Supported Languages . . . 44
2.4.17.2 References . . . 44
2.4.18 CyPhySim . . . 44
2.4.18.1 Supported Languages . . . 44
2.4.18.2 References . . . 44
2.4.19 DIPLODOCUS . . . 44
2.4.19.1 Supported Languages . . . 44
2.4.19.2 References . . . 44
2.4.20 Dymola . . . 44
2.4.20.1 Supported Languages . . . 45
2.4.20.2 References . . . 45
2.4.21 EBTresos . . . 45
2.4.21.1 Supported Languages . . . 45

2.4.21.2 References . . . 45
2.4.22 ESMoL . . . 45
2.4.22.1 Supported Languages . . . 45
2.4.22.2 References . . . 45
2.4.23 EclipseEDT . . . 45
2.4.23.1 Supported Languages . . . 45
2.4.23.2 References . . . 45
2.4.24 EclipseEMF . . . 45
2.4.24.1 Supported Languages . . . 45
2.4.24.2 References . . . 45
2.4.25 EclipseERD . . . 45
2.4.25.1 Supported Languages . . . 46
2.4.25.2 References . . . 46
2.4.26 EclipseEpsilon . . . 46
2.4.26.1 Supported Languages . . . 46
2.4.26.2 References . . . 46
2.4.27 FCM . . . 46
2.4.27.1 Supported Languages . . . 46
2.4.27.2 References . . . 46
2.4.28 FOMA . . . 46
2.4.28.1 Supported Languages . . . 46
2.4.28.2 References . . . 46
2.4.29 GEMOCStudio . . . 46
2.4.29.1 Supported Languages . . . 46
2.4.29.2 References . . . 46
2.4.30 IRM-SATool . . . 46
2.4.30.1 Supported Languages . . . 46
2.4.30.2 References . . . 47
2.4.31 Kronos . . . 47
2.4.31.1 Supported Languages . . . 47
2.4.31.2 References . . . 47
2.4.32 LTSA . . . 47
2.4.32.1 Supported Languages . . . 47
2.4.32.2 References . . . 47
2.4.33 MASSIF . . . 47
2.4.33.1 Supported Languages . . . 47
2.4.33.2 References . . . 47
2.4.34 MAST . . . 47

2.4.34.1 Supported Languages . . . 47
2.4.34.2 References . . . 47
2.4.35 MATSim . . . 47
2.4.35.1 Supported Languages . . . 48
2.4.35.2 References . . . 48
2.4.36 MoDeS . . . 48
2.4.36.1 Supported Languages . . . 48
2.4.36.2 References . . . 48
2.4.37 MyCCM-Hi . . . 48
2.4.37.1 Supported Languages . . . 48
2.4.37.2 References . . . 48
2.4.38 NuSMV . . . 48
2.4.38.1 Supported Languages . . . 48
2.4.38.2 References . . . 48
2.4.39 OBPExplorer . . . 48
2.4.39.1 Supported Languages . . . 48
2.4.39.2 References . . . 49
2.4.40 OMNet++ . . . 49
2.4.40.1 Supported Languages . . . 49
2.4.40.2 References . . . 49
2.4.41 OSATE . . . 49
2.4.41.1 Supported Languages . . . 49
2.4.41.2 References . . . 49
2.4.42 Ocarina . . . 49
2.4.42.1 Supported Languages . . . 49
2.4.42.2 References . . . 49
2.4.43 OpenModelica . . . 49
2.4.43.1 Supported Languages . . . 49
2.4.43.2 References . . . 49
2.4.44 Overture . . . 49
2.4.44.1 Supported Languages . . . 49
2.4.44.2 References . . . 50
2.4.45 PHAVer . . . 50
2.4.45.1 Supported Languages . . . 50
2.4.45.2 References . . . 50
2.4.46 PRISM . . . 50
2.4.46.1 Supported Languages . . . 50
2.4.46.2 References . . . 50

2.4.47 Palladio . . . 50
2.4.47.1 Supported Languages . . . 50
2.4.47.2 References . . . 50
2.4.48 Papyrus . . . 50
2.4.48.1 Supported Languages . . . 50
2.4.48.2 References . . . 50
2.4.49 ProCom . . . 50
2.4.49.1 Supported Languages . . . 51
2.4.49.2 References . . . 51
2.4.50 Ptolemy . . . 51
2.4.50.1 Supported Languages . . . 51
2.4.50.2 References . . . 51
2.4.51 ROS . . . 51
2.4.51.1 Supported Languages . . . 51
2.4.51.2 References . . . 51
2.4.52 RSA . . . 51
2.4.52.1 Supported Languages . . . 51
2.4.52.2 References . . . 51
2.4.53 Rainbow . . . 51
2.4.53.1 Supported Languages . . . 51
2.4.53.2 References . . . 52
2.4.54 Remes . . . 52
2.4.54.1 Supported Languages . . . 52
2.4.54.2 References . . . 52
2.4.55 SCADE . . . 52
2.4.55.1 Supported Languages . . . 52
2.4.55.2 References . . . 52
2.4.56 SOFA-HI . . . 52
2.4.56.1 Supported Languages . . . 52
2.4.56.2 References . . . 52
2.4.57 SPL . . . 52
2.4.57.1 Supported Languages . . . 53
2.4.57.2 References . . . 53
2.4.58 STOOD . . . 53
2.4.58.1 Supported Languages . . . 53
2.4.58.2 References . . . 53
2.4.59 Simulink . . . 53
2.4.59.1 Supported Languages . . . 53

2.4.59.2 References . . . 53
2.4.60 Spin . . . 53
2.4.60.1 Supported Languages . . . 53
2.4.60.2 References . . . 53
2.4.61 Stage . . . 53
2.4.61.1 Supported Languages . . . 53
2.4.61.2 References . . . 54
2.4.62 StrataGEM (Strategy Generic Extensible Modelchecker) . . . 54
2.4.62.1 Supported Languages . . . 54
2.4.62.2 References . . . 54
2.4.63 SyVoLT . . . 54
2.4.63.1 Supported Languages . . . 54
2.4.63.2 References . . . 54
2.4.64 SystemDesk . . . 54
2.4.64.1 Supported Languages . . . 54
2.4.64.2 References . . . 54
2.4.65 TINA_SELT (TIme petri Net Analyzer - State/Event LTL model checker) . . . 54
2.4.65.1 Supported Languages . . . 55
2.4.65.2 References . . . 55
2.4.66 TTool (tea-tool) . . . 55
2.4.66.1 Supported Languages . . . 55
2.4.66.2 References . . . 55
2.4.67 TURTLE . . . 55
2.4.67.1 Supported Languages . . . 55
2.4.67.2 References . . . 55
2.4.68 TargetLink . . . 55
2.4.68.1 Supported Languages . . . 55
2.4.68.2 References . . . 55
2.4.69 UMLAnalyzer . . . 55
2.4.69.1 Supported Languages . . . 55
2.4.69.2 References . . . 55
2.4.70 UMLMAST . . . 55
2.4.70.1 Supported Languages . . . 56
2.4.70.2 References . . . 56
2.4.71 UPPAAL . . . 56
2.4.71.1 Supported Languages . . . 56
2.4.71.2 References . . . 56
2.4.72 UppaalSMC (Statistical Model Checking Extension for the UPPAAL Toolset.) 56

2.4.72.1 Supported Languages 56
2.4.72.2 References 56
2.4.73 VIATRA 56
2.4.73.1 Supported Languages 56
2.4.73.2 References 57
2.4.74 Z3 57
2.4.74.1 Supported Languages 57
2.4.74.2 References 57
2.4.75 Zen-RUCM 57
2.4.75.1 Supported Languages 57
2.4.75.2 References 57
2.4.76 eC3M (Embedded Component Container Connector Middleware) 57
2.4.76.1 Supported Languages 57
2.4.76.2 References 57
2.4.77 jDEECo 57
2.4.77.1 Supported Languages 57
2.4.77.2 References 57

3 Glossary of Terms for Cyber Physical Systems 58

4 Summary and Future Work 60

Bibliography 61

1 Introduction

This report on the State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development of Working Group1 (WG1) on Foundations of the ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS) first presents a catalog of modeling languages and tools in chapter 2. Then a Glossary of Terms for Cyber Physical Systems is presented in chapter 3.

The work of WG1 revealed that the dependencies between the framework targeted in deliverable D1.2 Giese and Blouin (2016) and this report was much more tight than initially expected. To avoid capturing some content of this report in a redundant form in the ontologies of the framework, it was decided instead to include relevant information in the ontologies and extract it automatically from the ontologies for this report .

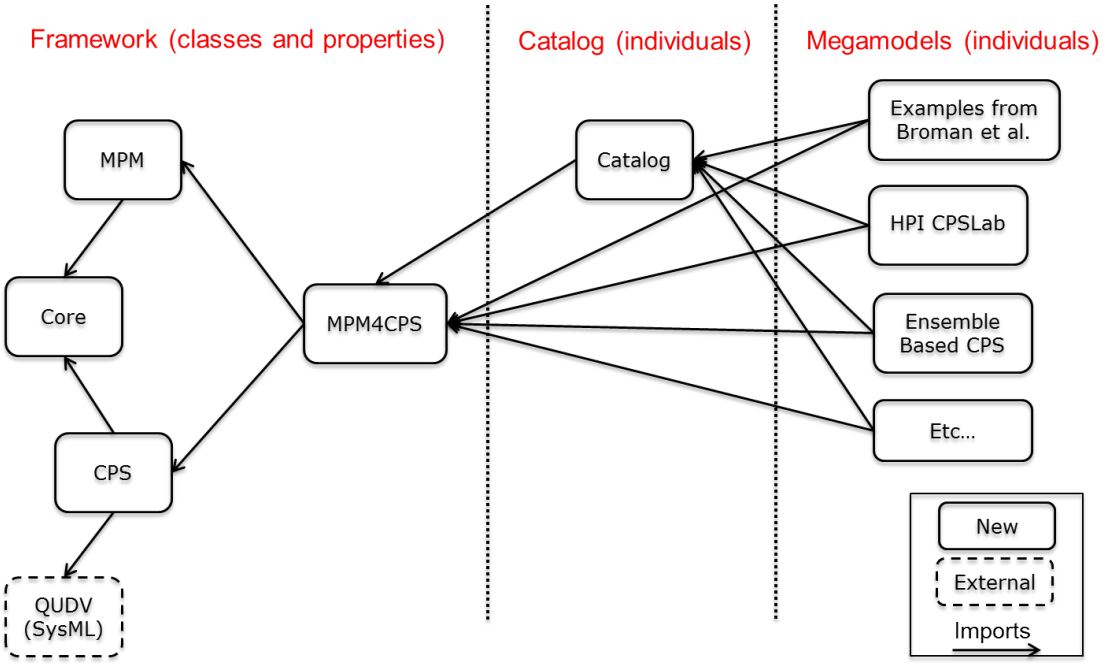


Figure 1.1: Overview of the structure of the MPM4CPS ontology

In figure 2.1 the structure of the framework and its elements in form of the different ontologies and their dependencies are presented. The figure also includes the data for the next two chapters of this report.

The first column depicts the framework and its ontologies as presented in the report on the Framework to Relate / Combine Modeling Languages and Techniques covered by deliverable D1.2 Giese and Blouin (2016). The Glossary of Terms for Cyber Physical Systems presented in chapter 3 of this report is extracted automatically from these ontologies and its contained concepts defining the framework.

In the second column the catalog of modeling languages and tools that consists of indiciduals of the MPM4CPS ontology as covered later in chapter 2 is presented. It is automatically derived from these individuals such that the ontology and its instances can be kept consistent with minimal coordination efforts.

In the third column, some examples for CPS development employing MPM in form of mega models are depicted. Those are presented in details in the report on the Framework to Relate / Combine Modeling Languages and Techniques covered by deliverable D1.2 Giese and Blouin

(2016). As depicted, these examples employ individuals of the catalog of languages and tools listed in this report in chapter 2 and instantiate the classes of the MPM4CPS ontology.

2 Structured Catalog of Modeling Languages and Tools

2.1 Introduction

A number of different tools is available that support multiparadigm modeling and/or CPS modeling, at different maturity level, designed by academia or industry. This richness is quite natural, for different reasons.

As first, CPS is a recent field, with no established standards and with different actors, each of which adopts a slightly different definition and a different approach for the topic: while multiparadigm modeling embraces a very wide spectrum of different topics, and actually has a very elastic and generic (informal) definition: the result is that tools were generated under different needs and from different directions, as the field spans from very theoretical, foundational approaches to very practical, application oriented approaches.

As second, this chaotic richness of tools is easily explained by the heterogeneity of users and backgrounds that are involved in the processes that are in the scope of interest of MPM4CPS that mirrors the intrinsic complexity of the systems that are in the scope of interest of MPM4CPS. They come from and work in different domains, each bringing the specialistic point of view, heritage and methods of their domain, they belong to different phases of the design, development and assessment cycle, or from different research fields, with different perspectives on the same problems and different views on the same 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.

In this chapter a catalogue of tools is presented, with a classification that is derived from the ontology. Some of the tools are released as commercial software in support of different applications, others are pure experimental tools devoted to support research in language engineering or model engineering. For each tool the main points have been resumed and classified, in order to provide a comprehensive support for candidate users and to guide them in exploring the offer to find the best match to their needs.

The structure of the catalog is inspired from the framework proposed by Broman et al. (2012) (see figure 1) where modeling languages and tools may be based on / support a set of formalisms and conversely, formalisms may be implemented by languages and tools. Therefore, the catalog proposed three top level lists for languages, tools and formalisms with subsections for referencing the related elements via the aforementioned relations. In addition, for each element of the catalog a set of references is provided.

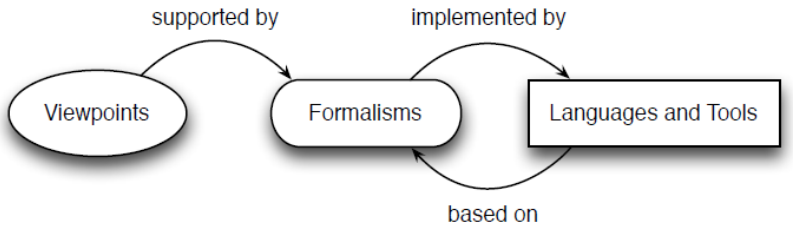


Figure 2.1: Framework for Viewpoints, Formalisms, Languages and Tools (from Broman et al. (2012))

2.2 Formalisms

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

2.2.1 AbstractStateMachines

Reviewer(s): Soumy The method built around the notion of Abstract State Machine (ASM) has been proved to be a scientifically well founded and an industrially viable method for the design and analysis of complex systems, which has been applied successfully to programming languages, protocols, embedded systems, architectures, requirements engineering, etc. The analysis covers both verification and validation, using mathematical reasoning (possibly theorem-prover-verified or model-checked) or experimental simulation (by running the executable models).

2.2.1.1 Implementing Languages

AbstractStateMachines is a formalism for the following languages:

- AsmL (see section 2.3.8)

2.2.1.2 References

http://web.eecs.umich.edu/gasm/intro.html

Borger (2005)

2.2.2 BayesianNetworks

Reviewer(s): Florin Leon

Bayesian networks (BNs), also known as belief networks (or Bayes nets for short), belong to the family of probabilistic graphical models (GMs). These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics.

2.2.2.1 Implementing Languages

None

2.2.2.2 References

Fuzzy Bayesian Networks: http://perso.telecom-paristech.fr/~chollet/Biblio/Articles/Domaines/BayesianNet/fuzzy-bayesian-networks-a.pdf

http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html

2.2.3 CTLSpecification

Reviewer(s): Didier Buchs, Soumy

e.g. A CTL specification is given as a formula in the temporal logic CTL.

2.2.3.1 Implementing Languages

CTLSpecification is a formalism for the following languages:

- CTL (see section 2.3.16)

2.2.3.2 References

2.2.4 CausalBlockDiagrams

Reviewer(s): Hans?

A Causal Block Diagram model is a graph made up of connected operation blocks. The connections stand for signals. Blocks can be purely algebraic such as Adder and Product, or may involve some notion of time such as Delay , Integrator and Derivative. Furthermore, Input and Output blocks are often used to model the system's connection to its environment.

2.2.4.1 Implementing Languages

None

2.2.4.2 References

2.2.5 CellularAutomata

Reviewer(s): Florin Leon

Cellular automata (CA) are an idealization of physical system in which space and time are discrete, and the physical quantities take into a finite set of values.

2.2.5.1 Implementing Languages

None

2.2.5.2 References

http://plato.stanford.edu/entries/cellular-automata/

2.2.6 DEECo

Reviewer(s): Rima? In scope of the ASCENS project have developed the DEECo component model (stands for Dependable Emergent Ensembles of Components) targeting design of systems consisting of autonomous, self-aware, and adaptable components. The components, implicitly organized in groups called ensembles, live in a very dynamic environment where a component can enter/exit an ensemble at any time. The goal of DEECo is to support development of applications in such a dynamic environment.

2.2.6.1 Implementing Languages

DEECo is a formalism for the following languages:

- DEECoDSL (see section 2.3.19)

2.2.6.2 References

2.2.7 DEECoSpecification

The main concepts of DEECo are heavily inspired by the concepts of the SCEL specification language. The main idea is to manage all the dynamism of the environment by externalizing the distributed communication between components to a component framework. The components access only local information and the distributed communication is performed implicitly by the framework. This way, the components have to be programmed as autonomous units, without relying on whether/how the distributed communication is performed, which makes them very robust and suitable for rapidly-changing environments.

2.2.7.1 Implementing Languages

None

2.2.7.2 References

2.2.8 DataFlow

Reviewer(s): Stefan, Etienne? Synchronous Dataflow: Synchronous Dataflow (SDF) is a model of computation first proposed by Edward A. Lee in 1986. It describes an application as a graph where a node represents a function (usually called actor) and an arc represents a communication channel between two functions. All the inputs of a function must be present to start its execution, and 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 characterized or extended into homogeneous data flow graphs, cyclo-static data flow graphs, scenario-aware data flow graphs, affine data flow graphs.

2.2.8.1 Implementing Languages

Lustre, Esterel and their integration in the SCADE tool suite 2.4.55. SIGNAL, and its Polychrony environment. Ptolemy. ZÃl'lus, bridging the GAP between SDF and ODEs (such as those implemented wit Mathlab/Simulink).

2.2.8.2 References

http://users.ece.utexas.edu/~bevans/courses/ee382c/lectures/08_sdf/

Kahn Process Networks: http://liacs.leidenuniv.nl/assets/Bachelorscripties/2011-20RoyKensmil.pdf

http://ptolemy.eecs.berkeley.edu/papers/02/synchronous/MurthyLee_MultidimensionalSDF.pdf

2.2.9 DataFlowTimed

Reviewer(s): Stefan

It's a Model of computation, not a formalism... perhaps we should move it to SystemC, but we don't have SystemC-AMS as language, so I'm not sure if we need this at all, if we don't describe SystemC-AMS

The Timed Data Flow (TDF) model of computation defined in the SystemC AMS 1.0 standard has already shown its value for signal-processing-oriented applications, such as RF communication and digital signal processing (DSP) systems, where the complex envelope of the modulated RF signal can be described as an equivalent baseband signal and where baseband algorithms are described naturally using data flow semantics. Because TDF is derived from the well-known Synchronous Data Flow (SDF) model of computation, high simulation performance can be obtained due to the calculation of a static schedule prior to simulation.

2.2.9.1 Implementing Languages

SystemC-AMS

2.2.9.2 References

http://www.accellera.org/images/downloads/standards/systemc/SystemC_AMS_2_0_LRM.pdf

http://www.accellera.org/resources/articles/amsspeed

http://leat.unice.fr/ECoFaC2012/presentations/Pecheux_part5.pdf

2.2.10 DifferentialEquations

Reviewer(s): Florin Leon, Eva

A differential equation is a mathematical equation containing functions and derivatives. Differential equations are in particular used for modeling the physical plant of a CPS. Differential equations can be broadly classified into ordinary differential equations (ODEs), differential-algebraic equations (DAEs), and partial differential equations (PDEs).

2.2.10.1 Implementing Languages

None

2.2.10.2 References

http://os.inf.tu-dresden.de/Studium/CPS/SS2015/02-Math.pdf

http://users.math.msu.edu/users/gnagy/teaching/ode.pdf

2.2.11 DiscreteEvent

Reviewer(s): Fernando Barros based on concurrent actors manipulating streams of timed events, such as DEVS, real-time process networks, or actor theories. The DE paradigm is prevalent in simulation frameworks in di erent application domains, from queueing systems and networks to circuits.

2.2.11.1 Implementing Languages

None

2.2.11.2 References

http://www.andrew.cmu.edu/user/dionisio/avicps2010-proceedings/an-adaptive-discrete-event-model-for-cyber-physical-systems.pdf

2.2.12 ElectricalLinearNetworks

Reviewer(s): Eva Navarro

The ELN formalism allows one to analyze/design electrical networks consisting of linear elements only (capacitor, inductor, resistor, current/voltage source, gyator, nullor, controlled current.voltage sources, etc.).

2.2.12.1 Implementing Languages

None

2.2.12.2 References

http://leat.unice.fr/ECoFaC2012/presentations/Pecheux_part5.pdf

http://www.accellera.org/resources/articles/amsspeed

2.2.13 EntityRelationship

Reviewer(s): Dominique «TODO: Provide rdfs:comment annotation assertion»

2.2.13.1 Implementing Languages

None

2.2.13.2 References

2.2.14 Fault Trees

Author(s): Mauro Iacono 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. Many variants have been proposed that extend the basic combinatorial nature of this formalism to include time, repairable components or repairing actions.

2.2.14.1 Implementing Languages

None

2.2.14.2 References

Ruijters and Stoelinga (2015)

2.2.15 FiniteStateProcess

Reviewer(s): Rima

«TODO: Provide rdfs:comment annotation assertion»

2.2.15.1 Implementing Languages

None

2.2.15.2 References

2.2.16 First Order Logic

Author(s): Moussa Amrani Reviewer(s): Florin Leon

First-Order Logic (abbreviated as FOL, and often referred to as 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).

2.2.16.1 Implementing Languages

FOL serves as the core formalisms for many tools, including:

- Theorem Provers like SMT_LIB (see section 2.3.59), Z3,

2.2.16.2 References

2.2.17 HyFlow (Hybrid Flow System Specification)

Reviewer(s): Fernando Barros, Eva Navaro

The Hybrid Flow Systems 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 stepsize differential equations solvers. HyFlow provides an integrative framework for combing models expressed in different modeling paradigms. In particular, fluid stochastic Petri-nets and geometric solvers, 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).

2.2.17.1 Implementing Languages

None

2.2.17.2 References

Barros (2003)

2.2.18 DiscontinuousSystems

Author(s): Eva Navarro

Discontinuous or non-smooth systems are a subclass of hybrid systems. They are several types depending on the type of discontinuity in the model representing the system. We have two main classes: 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.

2.2.18.1 Implementing Languages

None

2.2.18.2 References

2.2.19 HybridAutomata

Reviewer(s): Eva Navarro

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 representations of hybrid systems.

2.2.19.1 LinearHybridAutomata

Reviewer(s): Eva Navarro

Implementing Languages

None.

References

2.2.19.2 NonLinearHybridAutomata

Reviewer(s): Eva Navarro

Implementing Languages

None.

References

2.2.19.3 StochasticHybridAutomata

Reviewer(s): Eva Navarro

Implementing Languages

None.

References

2.2.19.4 TimedAutomata

Reviewer(s): Eva Navaro, Moussa Amrani

In automata theory, a timed automaton is a finite automaton extended with a finite set of real-valued clocks. During a run of a timed automaton, clock values increase all with the same speed. Along the transitions of the automaton, clock values can be compared to integers. These comparisons form guards that may enable or disable transitions and by doing so constrain the possible behaviors of the automaton. Further, clocks can be reset. Timed automata are a subclass of a type hybrid automata.

Implementing Languages

TimedAutomata is a formalism for the following languages:

- UPPAALRequirementSpecificationLanguage (see section 2.3.75)

References

2.2.19.5 TimeAutomataPriced (Priced/Probabilistic Timed Automata (PTAs))

Reviewer(s): Eva Navaro, Moussa Amrani

A priced timed automaton over X is an annotated directed graph with a distinguished vertex called the initial location. In the tradition of timed automata, we call vertices locations. An edge is decorated with a guard, an action and a reset set. We say that an edge is enabled if the guard evaluates to true and the source location is active. A reset set is a set of clocks. The intuition is that the clocks in the reset set are set to zero whenever the edge is taken.

Implementing Languages

TimeAutomataPriced is a formalism for the following languages:

- PRISMLanguage (see section 2.3.51)

References

2.2.19.6 TimedAutomataStochastic

Reviewer(s): Eva Navaro, Moussa Amrani

A stochastic timed automaton is a purely stochastic process defined on a timed automaton, in which both delays and discrete choices a are made randomly.

Implementing Languages

TimedAutomataStochastic is a formalism for the following languages:

- UPPAALSMCSpecificationLanguage (see section 2.3.76)

References

2.2.19.7 I/O_Automata

Reviewer(s): Paulo Careira

I/O automata provide an appropriate model for discrete event systems consisting of concurrently operating components.

Implementing Languages

None

References

Hybrid I/O Automata: http://groups.csail.mit.edu/tds/papers/Lynch/DIMACS95.pdf

Timed I/O Automata: http://people.cs.aau.dk/~adavid/ecdar/hscc10.pdf

2.2.19.8 Implementing Languages

HybridAutomata is a formalism for the following languages:

- StateFlow
- NuSMVLanguage (see section 2.3.47)

2.2.19.9 References

2.2.20 LabelledTransitionSystem

Reviewer(s): Didier Buchs To be removed

Labelled Transition Systems (LTS), are operational models of system behaviours that can be analysed in various ways with respect to given safety properties of the system. However, they often give a holistic view of the system, thereby also covering behaviour that is undesirable according to the system specification. It is therefore in the interest of requirements engineers to re ne these LTS models so that they can provide a more synthesised view of the system using scenarios and knowledge about the system domain.

2.2.20.1 Implementing Languages

None

2.2.20.2 References

TODO: LTSA developed in JAVA

2.2.21 LinearSignalFlow

Reviewer(s): Moussa Amrani «TODO: Provide rdfs:comment annotation assertion»

2.2.21.1 Implementing Languages

None

2.2.21.2 References

2.2.22 MarkovChains

A Markov process with finite or countable state space. The theory of Markov chains was created by A.A. Markov who, in 1907, initiated the study of sequences of dependent trials and related sums of random variables.

2.2.22.1 Implementing Languages

MarkovChains is a formalism for the following languages:

- PRISMLanguage (see section 2.3.51)

2.2.22.2 References

Continuous Markov Chains: http://www.columbia.edu/~ks20/stochastic-I/stochastic-I-CTMC.pdf

Markov Decision Process : https://www.cs.ubc.ca/~kevinlb/teaching/cs322%20-%202009-10/Lectures/DT3.pdf

Discrete Markov Chains: http://www.columbia.edu/~ks20/stochastic-I/stochastic-I-MCI.pdf

2.2.23 MessageDescriptionSpecification

Reviewer(s): Dominique Blouin The format of this language is simple: a message description is a list of data field descriptions and constant definitions on separate lines.

2.2.23.1 Implementing Languages

None

2.2.23.2 References

2.2.24 PetriNet

Didier Buchs, Mauro Iacono, Soumy

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems. Systems that are considered as discrete dynamic systems. 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, represented by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Some sources state that Petri nets were invented in August 1939 by Carl Adam Petri at the age of 13 for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, Business Process Model and Notation and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. There are attempts for all these formalisms to have semantics in term of translation into Petri nets. Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process 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.

2.2.24.1 Supported Extended Formalisms

Petri Nets have several extensions which are concretised in the following formalisms, it must be noted that the computing power can change according to these notations and then the analysability of the models:

- PetriNetPrioritised (Petri Net 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. (see section 2.2.27)

- PetriNetStochastic (Stochastic Petri Nets)(see section 2.2.28)
- PetriNetColoured (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 (see section 2.2.25)
- PetriNetTimed (Time Petri Net or Timed Petri Net)(see section 2.2.29) These formalisms introduce time as a modeling dimension 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.
- PetriNetDualistic (see section 2.2.26)

2.2.24.2 Supporting Tools

PetriNetLanguage and its variants is implemented by various tools, we can cite among the the following tools:

- Lola,
- TAPALL,
- GreatSPN,
- ORIS,
- Alpina, StrataGEM (see section 2.4.62)
- TINA_SELT (see section 2.4.65)

In order to evaluate the power of these tools benchmarks have been proposed and are anually evaluated in a competition called the modeling contest. **?**

2.2.24.3 References

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems.

2.2.24.4 Implementing Languages

None

2.2.24.5 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

2.2.25 PetriNetColoured

Reviewer(s): Didier Buchs, Soumy

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.

2.2.25.1 Implementing Languages

None

2.2.25.2 References

2.2.26 PetriNetDualistic

Reviewer(s): Didier Buchs, Soumy

Dualistic Petri nets (dPNs) are a process-class variant of Petri nets. Like Petri nets in general and many related formalisms and notations, they are used to describe and analyze process architecture.

2.2.26.1 Implementing Languages

None

2.2.26.2 References

2.2.27 PetriNetPrioritised

Reviewer(s): Didier Buchs, Soumy

A Prioritised Petri Net is a structure (PN, PF), where PN is a Petri Net and PF is a priority function that maps transitions into non-negative natural numbers representing their priority level.

2.2.27.1 Implementing Languages

None

2.2.27.2 References

2.2.28 PetriNetStochastic

Stochastic Petri nets are a form of Petri net where the transitions fire after a probabilistic delay determined by a random variable.

2.2.28.1 Implementing Languages

None

2.2.28.2 References

2.2.29 PetriNetTimed

Reviewer(s): Didier Buchs, Soumy

«TODO: Provide rdfs:comment annotation assertion»

2.2.29.1 Implementing Languages

PetriNetTimed is a formalism for the following languages:

- FIACRE (see section 2.3.32)

2.2.29.2 References

2.2.30 ProcessAlgebras

Reviewer(s): Florin Leon, Didier Buchs

An algebraic approach to the study of concurrent processes. Its tools are algebraical languages for the specification of processes and the formulation of statements about them, together with calculi for the verification of these statements.

2.2.30.1 Implementing Languages

None

2.2.30.2 References

2.2.31 TFPG (Timed Failure Propagation Graph)

Reviewer(s): Etienne Borde, Dominique Blouin, Rima Al Ali

A Timed Failure Propagation Graph (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 analyze 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.

2.2.31.1 Implementing Languages

None

2.2.31.2 References

http://www.mikand.net/data/smt_based_validation_of_timed_failure_propagation_graphs.pdf

2.2.31.3 Implementing Languages

TimedTransitionSystems is a formalism for the following languages:

- FIACRE (see section 2.3.32)

2.2.31.4 References

2.2.32 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 (noded 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.

2.2.32.1 Implementing Languages

None

2.2.32.2 References

2.3 Languages

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

2.3.1 AADL (Architecture Analysis and Design Language)

The AADL is designed for the specification, analysis, automated integration and code generation of real-time performance-critical (timing, safety, schedulability, fault tolerant, security, etc.) distributed computer systems. It provides a new vehicle to allow analysis of system designs (and system of systems) prior to development and supports a model-based, model-driven development approach throughout the system life cycle.

2.3.1.1 Supported Formalisms

None

2.3.1.2 Supporting Tools

AADL is implemented by the following tools:

- AADLInspector (see section 2.4.2)
- OSATE (see section 2.4.41)
- Ocarina (see section 2.4.42)

2.3.1.3 References

2.3.2 ACME (Architecture Description Interchange Language)

Reviewer(s): Miguel Goulão

There are several examples of Architectural Description Languages (ADLs), such as Aesop, Adage, C2, Darwin, Rapide, SADL, Unicon, MetaH, or Wright. While such ADLs considerably overlap on the core, each ADL focuses on different aspects of the software architecture and solving different categories of problems. This diversity raises difficulties in interchanging information among different ADLs. Providing an ADL which would support the features of all those ADLs would be impractical, and developing mappings among each pair of languages would require an excessive amount of effort. Acme was proposed as a generic language which can be used for expressing architectural concepts which are core to ADLs. The rationale was that rather than providing transformations between the different pairs of languages, Acme could be used as a common representation of architectural concepts to support the interchange of information reducing 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.

2.3.2.1 Supported Formalisms

Acme's elements are formally defined in a relational representation in a constraint logic language (see Wile (1996)).

2.3.2.2 Supporting Tools

ACME is implemented by the following tools:

- AcmeStudio (see section 2.4.9)

2.3.2.3 References

Garlan et al. (1997) Garlan et al. (2000) Wile (1996)

2.3.3 AML

ARC Macro Language: AML is a robust programming language that allows you access to a range of functionality, from automating common tasks in ARC/INFO, to creating complete GUI-based, multithreaded applications.

2.3.3.1 Supported Formalisms

None

2.3.3.2 Supporting Tools

AML is implemented by the following tools:

- ArcGIS (see section 2.4.12)

2.3.3.3 References

2.3.4 ATL

«TODO: Provide rdfs:comment annotation assertion»

2.3.4.1 Supported Formalisms

None

2.3.4.2 Supporting Tools

None

2.3.4.3 References

2.3.5 AUTOSARLanguage (AUTomotive Open System ARchitecture)

«TODO: Provide rdfs:comment annotation assertion»

2.3.5.1 Supported Formalisms

None

2.3.5.2 Supporting Tools

None

2.3.5.3 References

2.3.6 Alloy

Alloy is a language for describing structures and a tool for exploring them. It has been used in a wide range of applications from finding holes in security mechanisms to designing telephone switching networks.

2.3.6.1 Supported Formalisms

None

2.3.6.2 Supporting Tools

Alloy is implemented by the following tools:

- AlloyTool (see section 2.4.10)

2.3.6.3 References

2.3.7 Artisan

Support probabilities on the transitions and interruptible area

2.3.7.1 Supported Formalisms

None

2.3.7.2 Supporting Tools

None

2.3.7.3 References

2.3.8 AsmL (Abstract State Machine Language)

AsmL is an industrial-strength executable specification language. It can be used at any stage of the programming process: design, coding, or testing. It is fully integrated into the Microsoft .NET environment: AsmL models can interoperate with any other .NET assembly, no matter what source language it is written in. AsmL uses XML and Word for literate specifications.

2.3.8.1 Supported Formalisms

None

2.3.8.2 Supporting Tools

None

2.3.8.3 References

2.3.9 AsmetaL

AsmetaL is the language used for the ASMs.

2.3.9.1 Supported Formalisms

AsmetaL is based on the following formalisms:

- AbstractStateMachines (see section 2.2.1)

2.3.9.2 Supporting Tools

AsmetaL is implemented by the following tools:

- Asmeta (see section 2.4.13)

2.3.9.3 References

2.3.10 BlockDiagram

Block diagrams allow you to graphically represent the mathematical relationships between signals in a system. They are especially suited to model control systems. In 20-sim a large library of block diagram elements is available. The elements are displayed in the Editor by icons. You can create block diagram models by dragging the elements to the Editor and making the proper connections between the elements. 20-sim allows you to create user defined block diagram elements with an arbitrary number of input and output signals.

2.3.10.1 Supported Formalisms

None

2.3.10.2 Supporting Tools

BlockDiagram is implemented by the following tools:

- 20Sim (see section 2.4.1)

2.3.10.3 References

2.3.11 BondGraph

Bond graphs are a network-like description of physical systems in terms of ideal physical processes. With the bond graph method, the system characteristics are split-up into an (imaginary) set of separate elements. Each element describes an idealized physical process. To facilitate drawing of bond graphs, the common elements are denoted by special symbols.

2.3.11.1 Supported Formalisms

None

2.3.11.2 Supporting Tools

BondGraph is implemented by the following tools:

- 20Sim (see section 2.4.1)

2.3.11.3 References

2.3.12 C

«TODO: Provide rdfs:comment annotation assertion»

2.3.12.1 Supported Formalisms

None

2.3.12.2 Supporting Tools

None

2.3.12.3 References

2.3.13 C++

«TODO: Provide rdfs:comment annotation assertion»

2.3.13.1 Supported Formalisms

None

2.3.13.2 Supporting Tools

None

2.3.13.3 References

2.3.14 CCSL (Clock Constraint Specification Language)

CCSL usages: - A syntax to specify time semantics explicitly and formally - A language to express timed requirements

2.3.14.1 Supported Formalisms

None

2.3.14.2 Supporting Tools

None

2.3.14.3 References

2.3.15 CDL (Context Description Language)

CDL aims at formalizing the context with scenarios and temporal properties using property patterns. This DSML is based on UML 2. A CDL model describes, on the one hand, the context using activity and sequence diagrams and, on the other hand, the properties to be checked using property patterns. The originality of CDL is its ability to link each expressed property to a context diagram, i.e. a limited scope of the system behavior. allows contexts with scenarios and temporal properties using property patterns to be specified.

2.3.15.1 Supported Formalisms

None

2.3.15.2 Supporting Tools

CDL is implemented by the following tools:

- OBPExplorer (see section 2.4.39)
- TINA_SELT (see section 2.4.65)

2.3.15.3 References

2.3.16 CTL (Computation Tree Logic)

Computation tree logic (CTL) is a branching-time logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be an actual path that is realized. It is used in formal verification of software or hardware artifacts, typically by software applications known as model checkers which determine if a given artifact possesses safety or liveness properties. For example, CTL can specify that when some initial condition is satisfied (e.g., all program variables are positive or no cars on a highway straddle two lanes), then all possible executions of a program avoid some undesirable condition (e.g., dividing a number by zero or two cars colliding on a highway).

2.3.16.1 Supported Formalisms

CTL is based on the following formalisms:

- CTLSpecification (see section 2.2.3)

2.3.16.2 Supporting Tools

None

2.3.16.3 References

2.3.17 Clafer

Atlas Model Management Architecture

2.3.17.1 Supported Formalisms

None

2.3.17.2 Supporting Tools

None

2.3.17.3 References

2.3.18 CoCoME

The Common Component Modelling Example" Component-based software development (CBSD) has changed the current paradigm of software development. As systems become more and more complex, CBSD is to a greater extend applied in industry and plays a more and more important role in research. In order to leverage CBSD to build correct and dependable component-based systems, research has developed various formal and semi-formal component models. However, many of these component models like DisCComp, Fractal, Focus, or UML Extensions concentrate on different yet related aspects of component modelling. These are for instance communication issues or performance aspects. This hinders their validation for practical usage. Therefore, the main goal of the research seminar is to evaluate and compare the practical appliance of existing component models using a common component-based system as modelling example.

2.3.18.1 Supported Formalisms

None

2.3.18.2 Supporting Tools

None

2.3.18.3 References

2.3.19 DEECoDSL (Dependable Emergent Ensembles of Component-Domain Specific Language)

«TODO: Provide rdfs:comment annotation assertion»

2.3.19.1 Supported Formalisms

None

2.3.19.2 Supporting Tools

None

2.3.19.3 References

2.3.20 DSLTrans

«TODO: Provide rdfs:comment annotation assertion»

2.3.20.1 Supported Formalisms

None

2.3.20.2 Supporting Tools

DSLTrans is implemented by the following tools:

- SyVoLT (see section 2.4.63)

2.3.20.3 References

2.3.21 EAST-ADL

It is an Architecture Description Language (ADL) for automotive embedded systems, developed in several European research projects.

2.3.21.1 Supported Formalisms

None

2.3.21.2 Supporting Tools

None

2.3.21.3 References

2.3.22 ECL (Epsilon Comparison Language)

A rule-based language for discovering correspondences (matches) between elements of models of diverse metamodels.

2.3.22.1 Supported Formalisms

None

2.3.22.2 Supporting Tools

ECL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.22.3 References

2.3.23 EGL (Epsilon Generation Language)

A template-based model-to-text language for generating code, documentation and other textual artefacts from models.

2.3.23.1 Supported Formalisms

None

2.3.23.2 Supporting Tools

EGL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.23.3 References

2.3.24 EML (Epsilon Merging Language)

EML is a hybrid, rule-based language for merging homogeneous or heterogeneous models. As a merging language requires all the features of a transformation language (merging model A with an empty model into model B is equivalent to transforming A->B), EML reuses the syntax and semantics of ETL and extends it with concepts specific to model merging.

2.3.24.1 Supported Formalisms

None

2.3.24.2 Supporting Tools

EML is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.24.3 References

2.3.25 EOL (Epsilon Object Language)

An imperative model-oriented scripting language that combines the procedural style of Javascript with the powerful model querying capabilities of OCL.

2.3.25.1 Supported Formalisms

None

2.3.25.2 Supporting Tools

EOL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.25.3 References

2.3.26 ERD (Entity Relationship Diagram)

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases.

2.3.26.1 Supported Formalisms

None

2.3.26.2 Supporting Tools

ERD is implemented by the following tools:

- EclipseERD (see section 2.4.25)

2.3.26.3 References

2.3.27 ETL (Epsilon Transformation Language)

A rule-based model-to-model transformation language that supports transforming many input to many output models, rule inheritance,lazy and greedy rules.

2.3.27.1 Supported Formalisms

None

2.3.27.2 Supporting Tools

ETL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.27.3 References

2.3.28 EVL (Epsilon Validation Language)

A model validation language that supports both intra and inter-model consistency checking, and provides out-of-the-box integration with EMF & GMF editors.

2.3.28.1 Supported Formalisms

None

2.3.28.2 Supporting Tools

EVL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.28.3 References

2.3.29 EWL (Epsilon Wizard Language)

A language tailored for interactive in-place transformations on model elements selected by the user. EWL provides out-of-the-box integration with EMF & GMF editors

2.3.29.1 Supported Formalisms

None

2.3.29.2 Supporting Tools

EWL is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.29.3 References

2.3.30 EclipseEGL

EGL is a programming language conceptually similar to many common languages that have come before it. The language borrows concepts familiar to anyone using statically typed languages like Java, COBOL, C, etc. However, it borrows a concept from UML (Universal Modeling Language) that is not typically found in statically typed programming language ? the concept of Stereotype. In UML, stereotypes are used to tag UML elements with metadata (in this case, metadata refers to information about the UML element, for example, information about a UML element called a "class"). Constraints can be defined by stereotype definitions such that elements stereotyped by the given stereotype must adhere to the defined constraints of that stereotype. Stereotypes in UML are used as a lightweight mechanism to extend the standard modeling concepts. Stereotypes in EGL are essentially the same idea and are used to extend the basic EGL core language concepts.

2.3.30.1 Supported Formalisms

None

2.3.30.2 Supporting Tools

None

2.3.30.3 References

2.3.31 EpsilonFlock

Epsilon Flock is a model migration language built atop EOL, for updating models in response to metamodel changes. Flock provides a rule-based transformation language for specifying model migration strategies. A conservative copying algorithm automatically migrates model values and elements that are not affected by the metamodel changes.

2.3.31.1 Supported Formalisms

None

2.3.31.2 Supporting Tools

EpsilonFlock is implemented by the following tools:

- EclipseEpsilon (see section 2.4.26)

2.3.31.3 References

2.3.32 FIACRE

Fiacre stands for "Format Intermï£¡diaire pour les Architectures de Composants Rï£¡partis Embarquï£¡s", french for "Intermediate Format for the Embedded Distributed Component Architectures". Fiacre is a formally defined language for representing compositionaly both the behavioural and timing aspects of embedded and distributed systems for formal verification and simulation purposes.

2.3.32.1 Supported Formalisms

FIACRE is based on the following formalisms:

- PetriNet (see section 2.2.24)
- PetriNetTimed (see section 2.2.29)
- TimedTransitionSystems (see section **??**)

2.3.32.2 Supporting Tools

FIACRE is implemented by the following tools:

- OBPExplorer (see section 2.4.39)
- TINA_SELT (see section 2.4.65)

2.3.32.3 References

2.3.33 FUML (Foundational Subset for Executable UML Models)

This specification defines a subset of UML 2 and specifies foundational execution semantics for it. This subset will be referred to as Foundational UML or fUML. Conformance to this specification has two aspects: - Syntactic Conformance: A conforming model must be restricted to the abstract syntax subset defined for fUML. - Semantic Conformance: A conforming execution tool must provide execution semantics for a conforming model consistent with the semantics specified for fUML.

2.3.33.1 Supported Formalisms

FUML is based on the following formalisms:

- EntityRelationship (see section 2.2.13)

2.3.33.2 Supporting Tools

None

2.3.33.3 References

2.3.34 IRM

IRM is a method and a corresponding model that allows for designing software-intensive Cyber-Physical Systems (siCPS) with a focus on dependability aspects. IRM is tailored for systems consisting of ensembles of components (e.g. DEECo-based systems), and provides a way to refine high-level system invariants into low-level system obligations, or equivalently to trace low-level system obligations to their rationale at the requirements space.

2.3.34.1 Supported Formalisms

None

2.3.34.2 Supporting Tools

None

2.3.34.3 References

2.3.35 IRM-SA

IRM-SA is an extension to IRM that allows for introducing alternative decompositions in the design. Each branch in an alternative decomposition corresponds to a different situation in the environment (captured by one or more assumptions) that dictates a different design in order for the parent invariant to be preserved.

2.3.35.1 Supported Formalisms

None

2.3.35.2 Supporting Tools

IRM-SA is implemented by the following tools:

- IRM-SATool (see section 2.4.30)

2.3.35.3 References

2.3.36 IconicDiagrams

Iconic diagrams or components are the building blocks of physical systems. They allow you to enter models of physical systems graphically, similar to drawing an engineering scheme. In 20-sim a large library of iconic diagram elements is available. The elements are displayed in the Editor by icons which look like the corresponding parts of the ideal physical model. You can create models by dragging the elements to the Editor and making the proper connections between the elements.

2.3.36.1 Supported Formalisms

None

2.3.36.2 Supporting Tools

IconicDiagrams is implemented by the following tools:

- 20Sim (see section 2.4.1)

2.3.36.3 References

2.3.37 Java

«TODO: Provide rdfs:comment annotation assertion»

2.3.37.1 Supported Formalisms

None

2.3.37.2 Supporting Tools

None

2.3.37.3 References

2.3.38 LTL (Linear Temporal Logic)

In logic, linear temporal logic or linear-time temporal logic (LTL) is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true,

etc. It is a fragment of the more complex CTL*, which additionally allows branching time and quantifiers. Subsequently LTL is sometimes called propositional temporal logic, abbreviated PTL. Linear temporal logic (LTL) is a fragment of S1S.

2.3.38.1 Supported Formalisms

None

2.3.38.2 Supporting Tools

None

2.3.38.3 References

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

This specification of a UML profile adds capabilities to UML for model-driven development of Real Time and Embedded Systems (RTES). This extension, called the UML profile for MARTE (in short MARTE for Modeling and Analysis of Real-Time and Embedded systems), provides support for specification, design, and verification/validation stages. This new profile is intended to replace the existing UML Profile for Schedulability, Performance and Time (formal/03-09-01).

2.3.39.1 Supported Formalisms

MARTE is based on the following formalisms:

- EntityRelationship (see section 2.2.13)

2.3.39.2 Supporting Tools

None

2.3.39.3 References

2.3.40 MTL (Model to Text Language)

«TODO: Provide rdfs:comment annotation assertion»

2.3.40.1 Supported Formalisms

None

2.3.40.2 Supporting Tools

MTL is implemented by the following tools:

- Acceleo (see section 2.4.8)

2.3.40.3 References

2.3.41 MessagesDescriptionLanguage

ROS uses a simplified messages description language for describing the data values (aka messages) that ROS nodes publish. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages. Message descriptions are stored in .msg files in the msg/ subdirectory of a ROS package.

2.3.41.1 Supported Formalisms

MessagesDescriptionLanguage is based on the following formalisms:

- MessageDescriptionSpecification (see section 2.2.23)

2.3.41.2 Supporting Tools

MessagesDescriptionLanguage is implemented by the following tools:

- ROS (see section 2.4.51)

2.3.41.3 References

2.3.42 MetaH

Model tasks, subprograms, processes, and non-functional properties

2.3.42.1 Supported Formalisms

None

2.3.42.2 Supporting Tools

None

2.3.42.3 References

2.3.43 MoTiF

The Modular Timed Graph Transformation language (MoTif) allows to model and execute model transformations. On the one hand, it provides a graphical user interface for the description of the graph transformation rules in a declarative way and on the other hand, a modelling environment to define the control structure of the transformation.

2.3.43.1 Supported Formalisms

None

2.3.43.2 Supporting Tools

MoTiF is implemented by the following tools:

- AToM3 (see section 2.4.5)

2.3.43.3 References

2.3.44 Modelica

Modelica is an object-oriented, declarative, multi-domain modeling language for component-oriented modeling of complex systems, e.g., systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents.

2.3.44.1 Supported Formalisms

None

2.3.44.2 Supporting Tools

Modelica is implemented by the following tools:

- Dymola (see section 2.4.20)
- AMESim (see section 2.4.4)
- OpenModelica (see section 2.4.43)

2.3.44.3 References

2.3.45 ModelicaML

A UML Profile for Modelica Modelica Modeling Language (ModelicaML) is a graphical modeling language for the description of time-continuous and time-discrete/event-based system dynamics.

2.3.45.1 Supported Formalisms

None

2.3.45.2 Supporting Tools

None

2.3.45.3 References

2.3.46 NaturalLanguage

Such as plain text as we can see in use cases for example.

2.3.46.1 Supported Formalisms

None

2.3.46.2 Supporting Tools

None

2.3.46.3 References

2.3.47 NuSMVLanguage

«TODO: Provide rdfs:comment annotation assertion»

2.3.47.1 Supported Formalisms

None

2.3.47.2 Supporting Tools

None

2.3.47.3 References

2.3.48 OCL

Miguel Goulão

The Object Constraint Language (OCL) is a declarative language for describing rules that apply to Unified Modeling Language (UML) models developed at IBM and now part of the UML standard. Initially, OCL was only a formal specification language extension to UML. OCL may now be used with any Meta-Object Facility (MOF) Object Management Group (OMG) meta-model, including UML. The Object Constraint Language is a precise text language that provides constraint and object query expressions on any MOF model or meta-model that cannot otherwise be expressed by diagrammatic notation. OCL is a key component of the new OMG standard recommendation for transforming models, the Queries/Views/Transformations (QVT) specification.

2.3.48.1 Supported Formalisms

None

2.3.48.2 Supporting Tools

The following tools support OCL:

- Eclipse OCL https://projects.eclipse.org/projects/modeling.mdt.ocl
- Papyrus http://www.eclipse.org/papyrus/

None

2.3.48.3 References

https://www.omg.org/spec/OCL/2.4/PDF

2.3.49 OMEGA2

Extension of UML

2.3.49.1 Supported Formalisms

None

2.3.49.2 Supporting Tools

None

2.3.49.3 References

2.3.50 OSATE2

«TODO: Provide rdfs:comment annotation assertion»

2.3.50.1 Supported Formalisms

None

2.3.50.2 Supporting Tools

None

2.3.50.3 References

2.3.51 PRISMLanguage

In order to construct and analyse a model with PRISM, it must be specified in the PRISM language, a simple, state-based language, based on the Reactive Modules formalism of Alur and Henzinger. This is used for all of the types of model that PRISM supports: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs) and probabilistic timed automata (PTAs). For background material on these models, look at the pointers to resources on the PRISM web site.

2.3.51.1 Supported Formalisms

PRISMLanguage is based on the following formalisms:

- MarkovChains (see section 2.2.22)
- TimeAutomataPriced (see section **??**)

2.3.51.2 Supporting Tools

PRISMLanguage is implemented by the following tools:

- PRISM (see section 2.4.46)

2.3.51.3 References

2.3.52 ParallelAssignmentLanguage

The input language of NuSMV is designed to allow the description of finite state systems that range from completely synchronous to completely asynchronous. The NuSMV language (like the language of SMV) provides for modular hierarchical descriptions and for the definition of reusable components. The basic purpose of the NuSMV language is to describe (using expressions in propositional calculus) the transition relation of a finite Kripke structure. This provides a great deal of flexibility, but at the same time it can introduce danger of inconsistency.

2.3.52.1 Supported Formalisms

None

2.3.52.2 Supporting Tools

ParallelAssignmentLanguage is implemented by the following tools:

- NuSMV (see section 2.4.38)

2.3.52.3 References

2.3.53 PetriNetLanguage

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Some sources state that Petri nets were invented in August 1939 by Carl Adam Petri at the age of 13 for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, Business Process Model and Notation and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.

2.3.53.1 Supported Formalisms

PetriNetLanguage is based on the following formalisms:

- PetriNetPrioritised (see section 2.2.27)
- PetriNet (see section 2.2.24)
- PetriNetStochastic (see section 2.2.28)
- PetriNetColoured (see section 2.2.25)
- PetriNetTimed (see section 2.2.29)
- PetriNetDualistic (see section 2.2.26)

2.3.53.2 Supporting Tools

PetriNetLanguage is implemented by the following tools:

- StrataGEM (see section 2.4.62)

2.3.53.3 References

2.3.54 ProMoBox

«TODO: Provide rdfs:comment annotation assertion»

2.3.54.1 Supported Formalisms

None

2.3.54.2 Supporting Tools

None

2.3.54.3 References

2.3.55 Promela

PROMELA (Process or Protocol Meta Language) is a verification modeling language introduced by Gerard J. Holzmann. The language allows for the dynamic creation of concurrent processes to model, for example, distributed systems. In PROMELA models, communication via message channels can be defined to be synchronous (i.e., rendezvous), or asynchronous (i.e., buffered). PROMELA models can be analyzed with the SPIN model checker, to verify that the modeled system produces the desired behavior.

2.3.55.1 Supported Formalisms

None

2.3.55.2 Supporting Tools

Promela is implemented by the following tools:

- Spin (see section 2.4.60)

2.3.55.3 References

2.3.56 PtidyOS

«TODO: Provide rdfs:comment annotation assertion»

2.3.56.1 Supported Formalisms

None

2.3.56.2 Supporting Tools

None

2.3.56.3 References

2.3.57 QVT

«TODO: Provide rdfs:comment annotation assertion»

2.3.57.1 Supported Formalisms

None

2.3.57.2 Supporting Tools

None

2.3.57.3 References

2.3.58 Reo_Coordination_Language

a domain-specific language for programming and analyzing coordination protocols that compose individual processes into full systems

2.3.58.1 Supported Formalisms

None

2.3.58.2 Supporting Tools

None

2.3.58.3 References

2.3.59 SMT_LIB

This website provides access to the following main artifacts of the initiative. - Documents describing the SMT-LIB input/output language for SMT solvers and its semantics; - Specifications of background theories and logics; - A large library of input problems, or benchmarks, written in the SMT-LIB language. - Links to SMT solvers and related tools and utilities

2.3.59.1 Supported Formalisms

SMT_LIB is based on the following formalisms:

- FirstOrderLogic (see section 2.2.16)

2.3.59.2 Supporting Tools

SMT_LIB is implemented by the following tools:

- Z3 (see section 2.4.74)

2.3.59.3 References

2.3.60 STUML (Spatio-Temporal UML Statechart)

STUML statechart is an extension of 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.

2.3.60.1 Supported Formalisms

None

2.3.60.2 Supporting Tools

None

2.3.60.3 References

2.3.61 SimPL

Simpl is a tool for quickly and efficiently implementing domain-specific languages.

2.3.61.1 Supported Formalisms

None

2.3.61.2 Supporting Tools

None

2.3.61.3 References

2.3.62 SimulinkLanguage

«TODO: Provide rdfs:comment annotation assertion»

2.3.62.1 Supported Formalisms

SimulinkLanguage is based on the following formalisms:

- DifferentialEquations (see section 2.2.10)

2.3.62.2 Supporting Tools

SimulinkLanguage is implemented by the following tools:

- Simulink (see section 2.4.59)

2.3.62.3 References

2.3.63 Stitch

a language for expressing adaptation strategies

2.3.63.1 Supported Formalisms

None

2.3.63.2 Supporting Tools

Stitch is implemented by the following tools:

- Rainbow (see section 2.4.53)

2.3.63.3 References

2.3.64 SysML (Systems Modeling Language)

The Systems Modeling Language (SysML) is general purpose visual modeling language for systems engineering applications. SysML is defined as a dialect of the Unified Modeling Language (UML) standard, and supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities.

2.3.64.1 Supported Formalisms

None

2.3.64.2 Supporting Tools

SysML is implemented by the following tools:

- TTool (see section 2.4.66)

2.3.64.3 References

2.3.65 SystemCSpecification

Reviewer(s): Stefan Klikovits, Dominique Blouin

add to glossary

SystemC, an IEEE standardised 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).

add nice style for the ++ in C++

The use of as a basis provides SystemC with flexibility and adaptability and 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.

SystemC's pragmatic approach as internal dsl is reflected in the absence of a formal semantics. However, several proposals have been made such as **?** and **?**, and there exist several approaches to formally verify SystemC, as discussed in **?**.

2.3.65.1 Implementing Languages

SystemCSpecification is a formalism for the following languages:

- SystemC (see section 2.3.66)

2.3.65.2 References

@bookBlack:2005:SGU:1197604, address = Secaucus, NJ, USA, title = SystemC: From the Ground Up, isbn = 0-387-29240-3, publisher = Springer-Verlag New York, Inc., author = Black, David C. and Donovan, Jack and Bunton, Bill and Keist, Anna, year = 2010

@bookzhang2002microelectrofluidic, series = Nano- and Microscience, Engineering, Technology and Medicine, title = Microelectrofluidic Systems: Modeling and Simulation, isbn = 978-1-4200-4049-4, publisher = CRC Press, author = Zhang, T. and Chakrabarty, K. and Fair, R.B., year = 2002, lccn = 2002019344

@inproceedingsRuf2001TheSS, title=The simulation semantics of systemC, author=Jürgen Ruf and Dirk W. Hoffmann and Joachim Gerlach and Thomas Kropf and Wolfgang Rosenstiel and Wolfgang Müller, booktitle=DATE, year=2001

@inproceedingsSalemFormalsemanticssynchronous2003, title = Formal Semantics of Synchronous SystemC, doi = 10.1109/DATE.2003.1253637, booktitle = Automation and Test in Europe Conference and Exhibition 2003 Design, author = Salem, A., year = 2003, pages = 376-381

@inproceedingsDBLP:conf/syde/HerberG15, author = Paula Herber and Sabine Glesner, title = Verification of Embedded Real-time Systems, booktitle = 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, pages = 1–25, year = 2015, crossref = DBLP:conf/syde/2015, url = https://doi.org/10.1007/978-3-658-09994-7_1, doi = $10.1007/978-3-658-09994-7_1$, $timestamp = Thu, 15 Jun 2017 21:34:07 +0200$, $biburl = https://dblp.org/rec/bib/conf/syde/HerberG15$, $bibsource = dblpcomputersciencebibliography, ht$

2.3.66 SystemC

SystemC is a set of C++ classes and macros which provide an event-driven simulation interface (see also discrete event simulation). These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax. SystemC processes can communicate in a simulated real-time environment, using signals of all the datatypes offered by C++, some additional ones offered by the SystemC library, as well as user defined. In certain respects, SystemC deliberately mimics the hardware description languages VHDL and Verilog, but is more aptly described as a system-level modeling language.

2.3.66.1 Supported Formalisms

SystemC is based on the following formalisms:

- ElectricalLinearNetworks (see section 2.2.12)
- LinearSignalFlow (see section 2.2.21)
- DataFlowTimed (see section 2.2.9)
- SystemCSpecification (see section 2.3.65)

2.3.66.2 Supporting Tools

SystemC is implemented by the following tools:

- DIPLODOCUS (see section 2.4.19)

2.3.66.3 References

2.3.67 TCTL (Timed Computation Tree Logic)

Tt has Timed-constrained until

2.3.67.1 Supported Formalisms

None

2.3.67.2 Supporting Tools

None

2.3.67.3 References

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.

2.3.68.1 Supported Formalisms

None

2.3.68.2 Supporting Tools

TEPE is implemented by the following tools:

- AVATAR (see section 2.4.7)

2.3.68.3 References

2.3.69 TimedTransitionSystemLanguage

«TODO: Provide rdfs:comment annotation assertion»

2.3.69.1 Supported Formalisms

TimedTransitionSystemLanguage is based on the following formalisms:

- TimedTransitionSystems (see section **??**)

2.3.69.2 Supporting Tools

None

2.3.69.3 References

2.3.70 UML (Unified Modeling Language)

The OMG's Unified Modeling Language (UML) helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements.

2.3.70.1 Supported Formalisms

UML is based on the following formalisms:

- EntityRelationship (see section 2.2.13)

2.3.70.2 Supporting Tools

UML is implemented by the following tools:

- AVATAR (see section 2.4.7)
- UMLMAST (see section 2.4.70)
- Papyrus (see section 2.4.48)
- DIPLODOCUS (see section 2.4.19)
- TTool (see section 2.4.66)

2.3.70.3 References

2.3.71 UML-RT

«TODO: Provide rdfs:comment annotation assertion»

2.3.71.1 Supported Formalisms

None

2.3.71.2 Supporting Tools

None

2.3.71.3 References

2.3.72 UMLMARTE

«TODO: Provide rdfs:comment annotation assertion»

2.3.72.1 Supported Formalisms

None

2.3.72.2 Supporting Tools

None

2.3.72.3 References

2.3.73 UMLProfile

«TODO: Provide rdfs:comment annotation assertion»

2.3.73.1 Supported Formalisms

None

2.3.73.2 Supporting Tools

UMLProfile is implemented by the following tools:

- TTool (see section 2.4.66)

2.3.73.3 References

2.3.74 UMLSysML

«TODO: Provide rdfs:comment annotation assertion»

2.3.74.1 Supported Formalisms

None

2.3.74.2 Supporting Tools

None

2.3.74.3 References

2.3.75 UPPAALRequirementSpecificationLanguage

It describes the languages used when defining UPPAAL system models, and requirement specifications. -The System Description section describes the language used when defining a system model. -The Requirements Specification section describes the language used when specifying requirements on the system model. -The Expressions section describes the syntax for expressions in the two languages.

2.3.75.1 Supported Formalisms

UPPAALRequirementSpecificationLanguage is based on the following formalisms:

- TCTL (see section **??**)

2.3.75.2 Supporting Tools

UPPAALRequirementSpecificationLanguage is implemented by the following tools:

- UPPAAL (see section 2.4.71)

2.3.75.3 References

2.3.76 UPPAALSMCSpecificationLanguage

Defines the query formula with probability

2.3.76.1 Supported Formalisms

UPPAALSMCSpecificationLanguage is based on the following formalisms:

- TimedAutomataStochastic (see section **??**)

2.3.76.2 Supporting Tools

UPPAALSMCSpecificationLanguage is implemented by the following tools:

- UppaalSMC (see section 2.4.72)

2.3.76.3 References

2.3.77 VDM-SL

VDM models are expressed in a specification language (VDM-SL) that supports the description of data and functionality. Data are defined by means of types built using constructors that

define structured data and collections such as sets, sequences and mappings from basic values such as Booleans and numbers. These types are very abstract, allowing the user to add any relevant constraints as data type invariants. Functionality is defined in terms of operations over these data types. Operations can be defined implicitly by preconditions and postconditions that characterize their behavior, or explicitly by means of specific algorithms. An extension of VDM-SL, called VDM++, supports object-oriented structuring of models and permits direct modeling of concurrency.

2.3.77.1 Supported Formalisms

None

2.3.77.2 Supporting Tools

VDM-SL is implemented by the following tools:

- Overture (see section 2.4.44)
- Crescendo (see section 2.4.17)

2.3.77.3 References

2.3.78 Xtend

«TODO: Provide rdfs:comment annotation assertion»

2.3.78.1 Supported Formalisms

None

2.3.78.2 Supporting Tools

None

2.3.78.3 References

2.3.79 xtext

«TODO: Provide rdfs:comment annotation assertion»

2.3.79.1 Supported Formalisms

None

2.3.79.2 Supporting Tools

None

2.3.79.3 References

2.4 Tools

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

2.4.1 20Sim

20-sim is a modeling and simulation program for mechatronic systems. With 20-sim you can enter model graphically, similar to drawing an engineering scheme. With these models you can simulate and analyze the behavior of multi-domain dynamic systems and create control systems. You can even generate C-code and run this code on hardware for rapid prototyping and HIL-simulation.

2.4.1.1 Supported Languages

None

2.4.1.2 References

2.4.2 AADLInspector

AADL Inspector is a model processing framework for AADL. Its aim is to provide an easy to use and extensible tool to perform static and dynamic analysis of AADL architectures, and to easily connect any AADL compliant verification tool or code generator.

2.4.2.1 Supported Languages

None

2.4.2.2 References

2.4.3 AF3

AF3 is a powerful open-source (Apache License) tool to develop embedded systems using models from the requirements to the hardware architecture, passing by the design of the logical architecture, the deployment and the scheduling. AF3 provides advanced features to support the user ensuring the quality of his/her system:formal analyses, synthesis methods, space exploration visualization...

2.4.3.1 Supported Languages

None

2.4.3.2 References

2.4.4 AMESim (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.

2.4.4.1 Supported Languages

None

2.4.4.2 References

2.4.5 AToM3

AToM3 is a tool for multi-paradigm modelling under development at the Modelling, Simulation and Design Lab (MSDL) in the School of Computer Science of McGill University. It is developed in close collaboration with Prof. Juan de Lara of the School of Computer Science, Universidad Autï£¡noma de Madrid (UAM), Spain. AToM3 stands for “A Tool for Multi-formalism and Meta-Modelling”.

2.4.5.1 Supported Languages

None

2.4.5.2 References

2.4.6 AToMPM

AToMPM stands for "A Tool for Multi-Paradigm Modeling". It is a research framework from which you can generate domain-specific modeling web-based tools that run on the cloud. AToMPM is an open-source framework for designing DSML environements, performing model

transformations, and manipulating and managing models. It runs completely over the web, making it independent from any operating system, platform, or device it may execute on. AToMPM follows the philosophy of modeling everything explicitly, at the right level of abstraction(s), using the most appropriate formalism(s) and process(es), being completely modeled by itself.

2.4.6.1 Supported Languages

None

2.4.6.2 References

2.4.7 AVATAR (AVATAR stands for Automated Verification of reAl Time softwARe.)

AVATAR stands for Automated Verification of reAl Time softwARe.

AVATAR targets the modeling and formal verification of real-time embedded systems.

The AVATAR profile reuses eight of the SysML diagrams (Package diagrams are not supported). AVATAR supports the following methodological phases:

Requirement capture. Requirements and properties are structured using AVATAR Requirement Diagrams. At this step, properties are just defined with a specific label.

Assumption modeling. Assumptions of system may be captured with an assumption modeling diagram, based on a SysML requirement diagram.

System analysis. A system may be analyzed using usual UML diagrams, such as Use Case Diagrams, Interaction Overview Diagrams (Supported by UML2, not by SysML) and Sequence Diagrams.

System design. The system is designed in terms of communicating SysML blocks described in an AVATAR Block Diagram, and in terms of behaviors described with AVATAR State Machines.

Property modeling. The formal semantics of properties is defined within TEPE Parametric Diagrams (PDs). Since TEPE PDs involve elements defined in system design (e.g, a given integer attribute of a block), TEPE PDs may be defined only after a first system design has been performed.

Formal verification can be conducted over the system design, and for each testcase defined in the Requirement Diagram.

Code generation can finally be used to generate a fully executable code. The latter can be compiled and executed on the SoCLib prototyping platform directly from TTool, or executed on your local host if the latter supports gcc and POSIX.

2.4.7.1 Supported Languages

None

2.4.7.2 References

2.4.8 Acceleo

Acceleo is a pragmatic implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard.

2.4.8.1 Supported Languages

None

2.4.8.2 References

2.4.9 AcmeStudio

Reviewer(s): Miguel GoulÃčo

AcmeStudio is a customizable editing environment and visualization tool for software architectural designs based on the Acme architectural description language (ADL). With AcmeStudio, you can define new Acme families and customize the environment to work with those families by defining diagram styles. AcmeStudio is an adaptable front-end that may be used in a variety of modeling and analysis applications. AcmeStudio is implemented as a plugin for Eclipse environment, an open source Java Integrated Development Environment. Eclipse provides a plugin-environment allowing easy extensions of AcmeStudio with new analyses and functionality, and customization of new architectural environments tailored to a particular organization.

2.4.9.1 Supported Languages

AcmeStudio supports the following languages:

- Acme (see section 2.3.2)

2.4.9.2 References

http://acme.able.cs.cmu.edu/AcmeStudio/

2.4.10 AlloyTool

Alloyï£¡s tool, the Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model by generating sample structures, and to check properties of the model by generating counterexamples. Structures are displayed graphically, and their appearance can be customized for the domain at hand.

2.4.10.1 Supported Languages

None

2.4.10.2 References

2.4.11 AnyLogic

AnyLogic is the only simulation tool that supports all the most common simulation methodologies in place today: System Dynamics, Process-centric (AKA Discrete Event), and Agent Based modeling. The unique flexibility of the modeling language enables the user to capture the complexity and heterogeneity of business, economic and social systems to any desired level of detail. AnyLogicï£¡s graphical interface, tools, and library objects allow you to quickly model diverse areas such as manufacturing and logistics, business processes, human resources, consumer and patient behavior. The object-oriented model design paradigm supported by AnyLogic provides for modular, hierarchical, and incremental construction of large models. AnyLogic is a simulation software for the entire business lifecycle.

2.4.11.1 Supported Languages

None

2.4.11.2 References

2.4.12 ArcGIS

ArcGIS for Desktop is the key to realizing the advantage of location awareness. Collect and manage data, create professional maps, perform traditional and advanced spatial analysis, and solve real problems.

2.4.12.1 Supported Languages

None

2.4.12.2 References

2.4.13 Asmeta

This site is dedicated to the Abstract State Machine Metamodel (AsmM, in brief), a metamodel for the Abstract State Machines (ASMs) formal method developed by following the guidelines of the Model-Driven Engineering (MDE).

2.4.13.1 Supported Languages

None

2.4.13.2 References

2.4.14 CHESS

Distributed dependable real-time embedded software systems, like Satellite on board software, are becoming increasingly complex due to the demand for extended functionalities or the reuse of legacy code and components. Model-Driven Engineering (MDE) approaches are good solutions to help build such complex systems. Addressing domain specific modeling (like component description and interaction, real-time constraints, ...) while keeping the flexibility and generality offered by languages like UML is a challenge in a context where software must be qualified according to safety and reliability standards.

2.4.14.1 Supported Languages

None

2.4.14.2 References

2.4.15 COMSOL

Simulation Tool for Electrical, Mechanical, Fluid Flow, and Chemical Applications

2.4.15.1 Supported Languages

None

2.4.15.2 References

2.4.16 Capella

Much more than just yet another modelling tool, Capella is a model-based engineering solution that has been successfully deployed in a wide variety of industrial contexts. Based on a graphical modelling workbench, it provides systems, software and hardware architects with rich methodological guidance relying on Arcadia, a comprehensive model-based engineering method.

2.4.16.1 Supported Languages

None

2.4.16.2 References

2.4.17 Crescendo

The Crescendo Tool is an open-source tool originally developed in the EU DESTECS (Design Support and Tooling for Embedded Control Software) research project. The focus is on using co-simulation to design and modelling cyber-physical systems. The tool is based on the Overture platform and Controllab's 20-sim.

2.4.17.1 Supported Languages

None

2.4.17.2 References

2.4.18 CyPhySim

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

2.4.18.1 Supported Languages

None

2.4.18.2 References

2.4.19 DIPLODOCUS

DIPLODOCUS stands for DesIgn sPace exLoration based on fOrmal Description teChniques, Uml and SystemC.

Basically, DIPLODOCUS targets the partitioning of Systems-on-Chip. Partitioning a system means finding the best candidate software and hardware architecture for executing a set of functions. This selection of architecture is thus made according to given criteria, e.g., cost, power consumption, performance, etc..

DIPLODOCUS supports the Y-Chart approach, i.e., the partitioning is done as follows:

Application modeling: functions of the system are first modeled. Functions might later be software or hardware implemented.

Architecture modeling: candidate hardware architectures are modeled in terms of parametrized nodes: execution nodes (CPU, hardware accelerators), communication nodes (buses, bridges) and storage nodes (memories).

Mapping modeling: functions are mapped onto a given candidate architecture, i.e. functions are allocate dto either CPUs or hardware accelerators, and communication between functions are allocated to communication and storage nodes.

Moreover, DIPLODOCUS works at a high level of abstraction, and offers non-deterministic operators: this is thus very fast to model a first system, and evaluate different mappings for these functions. Moreover, all is graphical (UML). Last but not least, formal proofs and performance evaluation by simulation can be done at the push of a button, directly from UML diagrams.

2.4.19.1 Supported Languages

None

2.4.19.2 References

2.4.20 Dymola

Dymola is a physical modelling and simulation tool, used for model based design of complex engineering systems.

2.4.20.1 Supported Languages

None

2.4.20.2 References

2.4.21 EBTresos

«TODO: Provide rdfs:comment annotation assertion»

2.4.21.1 Supported Languages

None

2.4.21.2 References

2.4.22 ESMoL

Is a design language that includes the discrete-time subset of Simulink and Stateflow as sub-modeling languages, but it extends those with modeling constructs for componentization, platform modeling, and deployment modeling.

2.4.22.1 Supported Languages

None

2.4.22.2 References

2.4.23 EclipseEDT

It is tool for EGL EGL, originally developed by IBM, is a programming technology designed to meet the challenges of modern, multi-platform application development by providing a common language and programming model across languages, frameworks, and runtime platforms. The language borrows concepts familiar to anyone using statically typed languages like Java, COBOL, C, etc. However, it borrows the concept of Stereotype from UML (Universal Modeling Language) that is not typically found in statically typed programming languages.

In a nutshell, EGL is a higher-level, universal application development language.

2.4.23.1 Supported Languages

None

2.4.23.2 References

2.4.24 EclipseEMF

The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

2.4.24.1 Supported Languages

None

2.4.24.2 References

2.4.25 EclipseERD

a database design tool that provides graphical representation of database tables, their columns and inter-relationships.

2.4.25.1 Supported Languages

None

2.4.25.2 References

2.4.26 EclipseEpsilon

Epsilon is a family of languages and tools for code generation, model-to-model transformation, model validation, comparison, migration and refactoring that work out of the box with EMF and other types of models.

2.4.26.1 Supported Languages

None

2.4.26.2 References

2.4.27 FCM

Flexible Component Model : profile to annotate port, connectors, and components

2.4.27.1 Supported Languages

None

2.4.27.2 References

2.4.28 FOMA

The FOAM tool allows users to capture behaviour in use-cases using annotations and to verify various temporal constraints.

2.4.28.1 Supported Languages

None

2.4.28.2 References

TODO: add natural language

2.4.29 GEMOCStudio

The GEMOC Studio is an eclipse package that contains components supporting the GEMOC methodology for building and composing executable Domain-Specific Modeling Languages (DSMLs).

2.4.29.1 Supported Languages

None

2.4.29.2 References

2.4.30 IRM-SATool

The IRM-SA design tool is part of the IRM-SA toolchain and can be used to create IRM-SA models. It relies on Eclipse's EMF and GMF technologies and on Epsilon modeling languages for the tasks of model manipulation and model tranformation.

2.4.30.1 Supported Languages

None

2.4.30.2 References

2.4.31 Kronos

Kronos is a model-checker for timed automata, that can minimize the region graph of a timed automaton as described in. - KRONOS is a tool developed with the aim to verify complex real-time systems. - In KRONOS, components of real-time systems are modeled by timed automata and the correctness requirements are expressed in the real-time temporal logic TCTL. - KRONOS checks whether a timed automaton satisfies a TCTL-formula. - KRONOS is freely distributed to academic institutions for non-profit use.

2.4.31.1 Supported Languages

None

2.4.31.2 References

2.4.32 LTSA

Labelled Transition System Analyser: LTSA is a verification tool for concurrent systems. It mechanically checks that the specification of a concurrent system satisfies the properties required of its behaviour. In addition, LTSA supports specification animation to facilitate interactive exploration of system behaviour.

2.4.32.1 Supported Languages

None

2.4.32.2 References

2.4.33 MASSIF

Massif is Matlab Simulink Integration Framework for Eclipse. Its purpose is to convert Simulink models to Eclipse-EMF models, and vice versa. This guide introduces the main features of the software for end-users. It also contains illustrative screenshots in order to ease the learning process and to show configuration settings.

2.4.33.1 Supported Languages

None

2.4.33.2 References

2.4.34 MAST

MAST is an open-source suite of tools to perform schedulability analysis of real-time distributed systems that assesses a rich variety of timing requirements. Via sensitivity analysis, you will know how far or close the system is from meeting those requirements. MAST uses a versatile and composable input model for the real-time behavior of the modules and platforms that form your system.

2.4.34.1 Supported Languages

None

2.4.34.2 References

2.4.35 MATSim

MATSim is an open-source framework to implement large-scale agent-based transport simulations.

2.4.35.1 Supported Languages

None

2.4.35.2 References

2.4.36 MoDeS

«TODO: Provide rdfs:comment annotation assertion»

2.4.36.1 Supported Languages

None

2.4.36.2 References

2.4.37 MyCCM-Hi

«TODO: Provide rdfs:comment annotation assertion»

2.4.37.1 Supported Languages

None

2.4.37.2 References

2.4.38 NuSMV

NuSMV is a symbolic model checker developed as a joint project between: -The Embedded Systems Unit in the Center for Information Technology at FBK-IRST -The Model Checking group at Carnegie Mellon University , the Mechanized Reasoning Group at University of Genova -The Mechanized Reasoning Group at University of Trento. NuSMV is a reimplementation and extension of SMV, the first model checker based on BDDs. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs, as a core for custom verification tools, as a testbed for formal verification techniques, and applied to other research areas. NuSMV2, combines BDD-based model checking component that exploits the CUDD library developed by Fabio Somenzi at Colorado University and SAT-based model checking component that includes an RBC-based Bounded Model Checker, which can be connected to the Minisat SAT Solver and/or to the ZChaff SAT Solver. The University of Genova has contributed SIM, a state-of-the-art SAT solver used until version 2.5.0, and the RBC package use in the Bounded Model Checking algorithms.

2.4.38.1 Supported Languages

None

2.4.38.2 References

2.4.39 OBPExplorer

OBP is an implementation of a CDL language translation in terms of formal languages. It takes as input a CDL model and generates a set of FIACRE programs after contexts splitting. OBP leverages existing academic simulators and model checkers, as TINA [LAAS] or OBP-Explorer [ENSTA-Bretagne].

2.4.39.1 Supported Languages

None

2.4.39.2 References

2.4.40 OMNet++

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators.

2.4.40.1 Supported Languages

None

2.4.40.2 References

2.4.41 OSATE

Osate 2 is an open-source tool platform to support AADL v2. In January 2012 correction to a number of errata to AADL v2 have been approved.

2.4.41.1 Supported Languages

None

2.4.41.2 References

2.4.42 Ocarina

Ocarina is a stand-alone AADL model processor, written in Ada. It is distributed under the GPLv3 plus runtime exception.

2.4.42.1 Supported Languages

None

2.4.42.2 References

2.4.43 OpenModelica

OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization ï£¡ the Open Source Modelica Consortium (OSMC).

2.4.43.1 Supported Languages

None

2.4.43.2 References

2.4.44 Overture

The Overture tool (www.overturetool.org) represents the opening of these tools. This tool is build on top of the Eclipse platform and it support all the VDM dialects: VDM-SL, VDM++and VDM Real-Time (VDM-RT). Many different features are included but the emphasis is on validation of VDM models by interpretation of executable subsets. This also includes support for DE notation VDM-RT used inside the Crescendo tool. Users who are only interested in Discrete Event (DE) modelling using one or more of the VDM dialects should use this Overture tool.

2.4.44.1 Supported Languages

None

2.4.44.2 References

2.4.45 PHAVer

Polyhedral Hybrid Automaton Verifyer: PHAVer is a tool for verifying safety properies of hybrid systems. It stands out from other tools with the following features: -exact and robust arithmetic with unlimited precision, -on-the-fly over-approximation of piecewise affine dynamics -improved algorithms and termination heuristics -support for compositional and assume-guarantee reasoning.

2.4.45.1 Supported Languages

None

2.4.45.2 References

2.4.46 PRISM

PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. It has been used to analyse systems from many different application domains, including communication and multimedia protocols, randomised distributed algorithms, security protocols, biological systems and many others.

2.4.46.1 Supported Languages

None

2.4.46.2 References

2.4.47 Palladio

Palladio is a software architecture simulation approach which analyses your software at the model level for performance bottlenecks, scalability issues, reliability threats, and allows for a subsequent optimisation. Palladio requires neither buying expensive executions environments (servers, networks, or storage) nor fully implementing a software product. Construction rules are automatically checked by Palladio and thus allow optimal software architectures without costly trial-and-error-cycles. Like in other engineering disciplines, Palladio enables software engineers to construct software straight and in the right way.

2.4.47.1 Supported Languages

None

2.4.47.2 References

2.4.48 Papyrus

To address any specific domain, every part of Papyrus may be customized: UML profile, model explorer, diagram notation and style, properties views, palette and creation menus, and much more...

2.4.48.1 Supported Languages

None

2.4.48.2 References

2.4.49 ProCom

Progress Component Model

2.4.49.1 Supported Languages

None

2.4.49.2 References

2.4.50 Ptolemy

The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. A software system called Ptolemy II is being constructed in Java. The work is conducted in the Center for Hybrid and Embedded Software Systems (CHESS) in the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley. The project is directed by Prof. Edward Lee. The project is named after Claudius Ptolemaeus, the second century Greek astronomer, mathematician, and geographer.

2.4.50.1 Supported Languages

None

2.4.50.2 References

2.4.51 ROS

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

2.4.51.1 Supported Languages

None

2.4.51.2 References

2.4.52 RSA

IBM Rational Software Architect is an advanced and comprehensive application design, modeling and development tool for end-to-end software delivery. The latest version is updated with the latest in design and modeling technologies, comprehensive support for emerging technologies around BPMN2, SOA and Java Enterprise Edition 5, and delivers the best of breed tooling that integrates with IBM's application lifecycle management solutions.

2.4.52.1 Supported Languages

None

2.4.52.2 References

2.4.53 Rainbow

To reduce the cost and improve the reliability of making changes to complex systems, we are developing new technology supporting automated, dynamic system adaptation via architectural models, explicit representation of user tasks, and performance-oriented run-time gauges.

2.4.53.1 Supported Languages

None

2.4.53.2 References

2.4.54 Remes

Remes is a tool for formal modeling of embedded resources such as storage, energy, communication, and computation. The model is a state-machine based behavioral language with support for hierarchical modeling, resource annotations, continuous time, and notions of explicit entry and exit points that make it suitable for component-based modeling of embedded systems. The analysis of RE ME S-based systems is centered around a weighted sum in which the variables represent the amounts of consumed resources.

2.4.54.1 Supported Languages

None

2.4.54.2 References

Moussa Amrani

2.4.55 SCADE

SCADE Suite is a product line of the ANSYS Embedded software family of products and solutions that empowers users with a Model-Based Development Environment for critical embedded software. With native integration of the formally-defined Scade language, SCADE Suite is the integrated design environment for critical applications spanning requirements management, model-based design, simulation, verification, qualifiable/certified code generation, and interoperability with other development tools and platforms.

2.4.55.1 Supported Languages

None

2.4.55.2 References

2.4.56 SOFA-HI

SOFA HI is an extension of the SOFA 2 component model, targeted at high-integrity real-time embedded systems.

The key additions and differences of SOFA HI comparing to SOFA 2 include various restrictions of the component model in order to make it more predictable and lightweight. For instance, SOFA HI restricts dynamic architecture reconfigurations to dynamic component updates at runtime only, while SOFA 2 supports more types of dynamic architecture reconfigurations). In addition, while SOFA 2 does not consider any restricted computational model, SOFA HI considers the Ravenscar Computational Model for local deployments, with an extension for distributed deployments.

2.4.56.1 Supported Languages

None

2.4.56.2 References

2.4.57 SPL

Stochastic Performance Logic Compared to functional unit testing, performance unit testing is more difficult, partially because correctness criteria are more difficult to express for performance than for functionality. Using the Stochastic Performance Logic (SPL), we aim to express assertions on code performance in relative, hardware-independent terms. Using the performance unit testing tools, these assertions can be automatically validated. Besides performance

unit testing, we experiment with using the performance unit tests to generate data for software documentation extended with performance information. Other research includes incorporating performance awareness into adaptive applications.

2.4.57.1 Supported Languages

None

2.4.57.2 References

2.4.58 STOOD

STOOD is a Software design tool that complies to both AADL and HOOD standards. AADL models can be defined to specify the complete host system of the applicative Software. Each identified AADL Process can then be refined down to target source code thanks to the HOOD detailed design process.

2.4.58.1 Supported Languages

None

2.4.58.2 References

2.4.59 Simulink

Simulink is a block diagram environment for multidomain simulation and Model-Based Design. It supports simulation, automatic code generation, and continuous test and verification of embedded systems.

Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

2.4.59.1 Supported Languages

None

2.4.59.2 References

2.4.60 Spin

Spin is a popular open-source software verification tool, used by thousands of people worldwide. The tool can be used for the formal verification of multi-threaded software applications.

2.4.60.1 Supported Languages

None

2.4.60.2 References

2.4.61 Stage

Stage (ver 3.0.0 and above) is equipped to work as standalone, wherein controllers integrated into stage enables motion, behaviour and processes for the robot(s). This tutorial discusses two such stage controllers.

2.4.61.1 Supported Languages

None

2.4.61.2 References

2.4.62 StrataGEM (Strategy Generic Extensible Modelchecker)

Strategy Generic Extensible Modelchecker (StrataGEM), a tool aimed at the analysis of Petri nets and other models of concurrency by means of symbolic model-checking techniques. StrataGEM marries the well know concepts of Term Rewriting (TR) to the efficiency of Decision Diagrams (DDs). TR systems are a great way to describe the semantics of a system, being readable and compact, but their direct implementation tends to be rather slow on large sets of terms. On the other hand, DDs have demonstrated their efficiency for model-checking, but translating a system semantics into efficient DDs operations is an expert's matter. StrataGEM describes the semantics of a system in terms of strategies over a TR system, and automatically translates these rules into operations on DD to handle the model-checking. The ultimate goal of StrataGEM is to become a verification framework for the different variants of Petri nets by separating the semantics of the model from the computation that performs model-checking.

2.4.62.1 Supported Languages

None

2.4.62.2 References

2.4.63 SyVoLT

SyVOLT ((Symbolic Verifier of mOdeL Transformations), a plugin for the Eclipse development environment for the verification of structural pre-/post-condition contracts on model transformations. The plugin allows the user to build transformations in our transformation language DSLTrans using a visual editor. The pre-/post-condition contracts to be proved on the transformation can also be built in a similar interface.

2.4.63.1 Supported Languages

None

2.4.63.2 References

2.4.64 SystemDesk

System Architecture Software SystemDesk is a system architecture tool that provides sophisticated and extensive support for modeling AUTOSAR architectures and systems for application software. Additionally, SystemDesk generates virtual ECUs (V-ECUs) out of the application software. The V-ECUs can be used as units under test with the dSPACE simulation platforms, such as the PC-based simulation platform VEOS for validating the ECU software.

2.4.64.1 Supported Languages

None

2.4.64.2 References

2.4.65 TINA_SELT (TIme petri Net Analyzer - State/Event LTL model checker)

Tina (TIme Petri Net Analyser)1 is a software environment for the editing and analysis of Petri net and Time Petri net (Merlin and Farber 1976). In addition to the usual editing and analysis facilities of such environments (computation of marking reachability sets, coverability trees, semi-flows), Tina offers various abstract state space constructions that preserve specific classes of properties of the concrete state spaces of the nets. These classes of properties may be general properties (reachability properties, deadlock freeness, liveness), specific properties relying on the linear structure of the concrete space state (linear time temporal logic properties, test

equivalence), or properties relying on its branching structure (branching time temporal logic properties, bisimulation).

2.4.65.1 Supported Languages

None

2.4.65.2 References

2.4.66 TTool (tea-tool)

TTool (pronounced "tea-tool") is a toolkit dedicated to the edition of UML and SysML diagrams, and to the simulation and formal validation of those diagrams. TTool supports several UML profiles, including: - DIPLODOCUS: UML profile dedicated to the partitioning of Systems-on-Chip or embedded systems. - AVATAR: SysML-based environment for the modeling and formal verification of real-time embedded software.. - SysML-Sec: SysML-based environment for the modeling and formal verification of real-time embedded systems with security and safety issues

2.4.66.1 Supported Languages

None

2.4.66.2 References

2.4.67 TURTLE

Verification and Simulation of TURTLE (Real-Time UML) Diagrams

2.4.67.1 Supported Languages

None

2.4.67.2 References

2.4.68 TargetLink

Automatic production code generator TargetLink is a software system that generates production code (C code) straight from the MATLAB/Simulink/Stateflow graphical development environment.

2.4.68.1 Supported Languages

None

2.4.68.2 References

2.4.69 UMLAnalyzer

«TODO: Provide rdfs:comment annotation assertion»

2.4.69.1 Supported Languages

None

2.4.69.2 References

2.4.70 UMLMAST

UML-MAST is a metodology and a set of tools for modeling and analizing object oriented real-time systems expressed in UML. It is based on the concept of the "Mast RT View" of the system, which describes in a qualitative and quantitative way the timing behavior, the real-time performance constraints and relevant implementation parameters from the real-time perspective.

The use of a real-time view allows the designer building the real-time system model gradually according to the evolution of the development process, feeding the analysis tools, and bringing back into the model the relevant timing responses. Therefore UML-MAST follows the model processing paradigm.

2.4.70.1 Supported Languages

None

2.4.70.2 References

2.4.71 UPPAAL

UPPAAL is an integrated tool environment for modeling, simulation and, verification of real-time embedded systems. Typical application areas of UPPAAL includes real-time controllers and communication protocols in particular, those where timing aspects are critical. Key features of UPPAAL v.4 : - A graphical system editor allowing graphical descriptions of systems. - A graphical simulator which provides graphical visualization. - A requirement specification editor. - A model-checker for automatic verification. - Generation of diagnostic trace

2.4.71.1 Supported Languages

None

2.4.71.2 References

2.4.72 UppaalSMC (Statistical Model Checking Extension for the UPPAAL Toolset.)

Statistical Model Checking (SMC) refers to a series of techniques that monitor several runs of the system with respect to some property, and then use results from the statistics to get an overall estimate of the correctness of the design. The approach has been applied to problems that are far beyond the scope of existing model checkers. In fact, SMC gets widely accepted in various research areas such as systems biology or software engineering, in particular for industrial applications. There are several reasons for this success. First, it is very simple to implement, understand and use (especially by industry, software engineers, and generally all people that are not pure researchers but customers for our results and tools). Second, it requires little or no extra modeling or specification effort, but simply an operational model of the system that can be simulated and checked against properties. Third, the use of Statistics allows to approximate undecidable problems. Finally, it is possible to easily distribute SMC.

2.4.72.1 Supported Languages

None

2.4.72.2 References

2.4.73 VIATRA

VIATRA: An Event-driven and Reactive Model Transformation Platform The VIATRA framework supports the development of model transformations with specific focus on event-driven, reactive transformations and offers a language to define transformations and a reactive transformation engine to execute certain transformations upon changes in the underlying model. Furthermore, the underlying incremental query engine, originating from the EMF-IncQuery project is reusable in different scenarios not related to model transformations.

2.4.73.1 Supported Languages

None

2.4.73.2 References

2.4.74 Z3

Z3 is a low level tool. It is best used as a component in the context of other tools that require solving logical formulas.

2.4.74.1 Supported Languages

None

2.4.74.2 References

2.4.75 Zen-RUCM

Zen-RUCM, built on top of RUCM, aims to tackle the challenges of requirement specification and analysis in different application domains (e.g., real-time systems, distributed systems, communication systems) and from various requirement specification concerns (e.g., variability, Non-Functional Requirements (NFR), crosscutting concerns).

2.4.75.1 Supported Languages

None

2.4.75.2 References

2.4.76 eC3M (Embedded Component Container Connector Middleware)

eC3M (pronounce: e triple-C M) is a component based modeling / middleware approach that is suitable for embedded and real-time applications. The application modeling is based on the Flex-eWare component model (FCM). This component model is aligned with the OMG standard (D&C), click here for more information on the component model.

2.4.76.1 Supported Languages

None

2.4.76.2 References

2.4.77 jDEECo

JDEECo framework is a prototype in java that illustrates basic DEECo concepts.

2.4.77.1 Supported Languages

None

2.4.77.2 References

3 Glossary of Terms for Cyber Physical Systems

For the glossary we were able to identify a list of terms presented in the two following Tables 3.1 and 3.2 that are important for MPM4CPS. As many terms are related or even included in the ontology already, we plan to integrate them into the ontology model in the long run rather the maintaining a separate source/representation, and generate this part of the document automatically.

term	subterm	class	area of origion
			dynamical systems theory
		types of systems	
Heterogeneous systems			
Multi-modal systems			
Multi-controller systems			
Logic-based switching systems			
Discrete-event systems			
Transition systems			
Variable structure systems			
Discontinuous/switched/non-smooth systems			
Complementarity systems			
Reset systems			
Jump systems			
Piecewise-affine systems			
Mixed logical dynamical systems			
Impulsive systems			
Cyber-physical systems of systems			
Cyber-physical networked embedded systems			
Large-scale smart systems			
		modelling	
Hybrid automata			
Bond graphs			
Petri nets			
Complementarity models			
Event-flow formulae			
Bisimulations			
Symbolic dynamics			
		analysis	
Sliding motions			
abstraction of energy in the system:			
	passivity		
	dissipativity		
Stability			
State space			
State partition			
			Verification
		verification properties	
Safety properties			
Liveness properties			
Deadnesss properties			
Reachability			
Correctness/consistency			
		verification techniques	
Model checking			
Theorem proving (deductive approach)			
Falsification			
Constraint satisfaction			
Boolean satisfiability			
Satisfiability modulo theories			
Symbolic methods			
Dynamically-aware verification			
automated reasoning			
		Validation	
Testing			
Debugging			
			Control
		control properties	
Controllability			
Observability			
Robustness			
Practical stability			
		control strategies	
Operation modes			
Adaptive control			
Decentralized/hierarchical control			
Optimal control			
		control concepts	
Feedback/feedforward			
Closed loops			
Environment			
Sensors/actuators			
Set points/control goals			
Disturbances/perturbations			
Parameter identification			
Signal processing			
Contingency/risk analysis			

Table 3.1: Terms for the glossary (1/2)

term	subterm	class	area of origion
			Behavior
		dynamical behaviours	
Chattering			
Zeno behaviour			
Sliding behaviour			
Discontinuous/sliding bifurcations (border collision, corner bifurcations)			
Oscillations/limit cycles			
Strange/complex behaviours (chaos)			
		Other behaviors	
Robustness			
Resilience			
Stability (different types)			
Emergent behaviour			
Phase transitions			
Collective/group behaviour			
Self-organization			
Adaptive networks			
Collaboration:			
	Cooperation		
	Competition		
	Swarming		
	Flocking		
	Consensus		
	Synchronization		
	Pinning control		
	Coupling		
		Communication:	
Agent-based systems			
Swarm intelligence			
		Topology:	
Graph theory			
Statistical physics			
		application domains	
systems biology			
financial markets			
social systems			
etc.			
		Else	
Observability/Visibility (black/grey/white box)			
Syntax vs. Semantics			
Visual vs. textual (or mixture) syntax			
System vs. Environment (vs. controller)			
Cognitive gap to the domain			
Suitability			
Intent (general): ilities			
Intent (of building models):			
Intent -> Objectives			
Validation vs. verification			
Tolerance			
Generality (general / domain-specific)			
Domain			
Properties that can be modeled (~ requirements)			
requirements vs. design languages			
Abstraction levels			
Abstraction: available info vs. questions			
Heterogeneity of ...			
Megamodel			
Autonomy			
Emerging Properties			
Unification			
Non-functior usability)			
User in a CPS			
Consistency			
Composability			
Semantic adaptation			
System of Systems			
Inductive			
planning process of modeling			
Variability			
Sensitivity			
Architecture			
Deployment			
X-in-the-loop			
Ontological vs. Linguistic (and consistency between them)			
Consistency (wrt. set of properties)			
System of Systems			
Smart objects			
Planning at system level			
governance			
technical control			
choreography			
orchestration			

Table 3.2: Terms for the glossary (2/2)



4 Summary and Future Work

In this report on the State-of-the-art on Current Formalisms used in Cyber-Physical Systems Development of Working Group1 (WG1) on Foundations of the ICT COST Action IC1404 Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS), we first presented a catalog of languages, formalisms, and tools in chapter 2. Then a glossary of terms for Cyber Physical Systems has been presented in chapter 3.

Both the catalog and glossary still need to be improved as future work. For the catalog, the documentation of each entry and the completeness of the catalog still have to be improved. In addition, the three main categories of languages, formalisms and tools of the catalog better populated, reviewed, and updated to reflect the finer classification provided by the ontology from which they are generated. Finally, the presented glossary of terms for Cyber-Physical Systems is currently still incomplete and has to be reworked to cover all relevant terms and be better linked to the ontologies.



Bibliography

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

Borger, E. (2005), Abstract state machines and high-level system design and analysis, *Theoretical Computer Science*, **336**, 2, pp. 205 – 207, ISSN 0304-3975.

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

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.

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).

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

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

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