Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs

INTO-CPS

Techniques for abstraction of continuous-time models into finite discrete-event models for model-checking

Technical Note Number: D5.1c

Version: 0.3

Date: 2015

Public Document

http://into-cps.au.dk

Contributors:

Jörg Brauer, VSI
Oliver Möller, VSI

Editors:

Jörg Brauer, VSI

Reviewers:

Francois Hantry, CLE
Imran Quadri, ST
Claes Dühring Jaeger, AI

Consortium:

Aarhus University	AU	Newcastle University	UNEW
University of York	UY	Linköping University	LIU
Verified Systems International GmbH	VSI	Controllab Products	CLP
ClearSy	CLE	TWT GmbH	TWT
Agro Intelligence	AI	United Technologies	UTRC
Softeam	ST		

Document History

Ver	Date	Author	Description
0.1	01-06-2015	Jörg Brauer	Initial document version
0.2	11-18-2015	Jörg Brauer	Ready for interval review
0.3	12-14-2015	Jörg Brauer	Reworked according to interval review comments

Abstract

This deliverable describes and discusses several ideas for abstraction mechanisms, which will be developed and implemented in the INTO-CPS project so as to support verification of co-simulation environments consisting of both, continuous time and discrete event models. The bounded model checking functionality implemented in the RT-Tester Model-Based Test Case Generator will be extended so that it supports such kinds of heterogenous co-simulation environments.

Contents

1 Introduction

Model checking of discrete event (DE) models has been extensively studied over the past four decades. Brilliant ideas paired with careful engineering have led to the invention of techniques such as symbolic model checking, SMT solving, bounded model checking (BMC) etc.; these techniques have made the verification of real-world systems a realistic yet still challenging task. However, model checking techniques for continuous time (CT) systems have not yet reached the maturity of techniques related to DE, most importantly with respect to scalability and tractability.

For real-world applications — the kind of applications the INTO-CPS project focuses on — it is thus advisable to abstract CT models into DE models for the purpose of model checking. Given a concrete CT model $\mathcal{M}_{CT}$, the key idea of abstraction in this context is to derive a map α such that $\mathcal{M}_{CT} \models \alpha(\mathcal{M}_{CT})$. Then, all behavior described by $\mathcal{M}_{CT}$ is as well present in $\alpha(\mathcal{M}_{CT})$, although α may introduce additional behavior. If checking properties on the abstract system $\alpha(\mathcal{M}_{CT})$, abstraction thus preserves soundness but comes at the cost of completeness, which means that model checking may yield false positive warnings.

This deliverable discusses how the abstraction α from a CT system to a DE system can be implemented so as to apply DE model checking techniques to systems that were originally designed as CT systems. Importantly, the document discusses how the abstraction can be modelled by means of timed state charts, which then allows the standard model checking techniques of the RT-Tester Model-Based Test Case Generator (RTT-MBT) to be applied.

1.1 Setting

The overall setting of the techniques discussed in this deliverable is discussed by means of the example given in Fig. 1. For this example, we do not focus on what the models do, but more on their connection. The overall configuration consists of three models, two of which are defined as DE models. The signal `voltage` computed by the DE model called *Environment* flows into the remaining two blocks, both of which use this value to compute their outputs. The express aim of the techniques discussed in this deliverable is to replace the CT component in Fig. 1 by an over-approximate DE component that soundly mimicks the behavior of the original model.

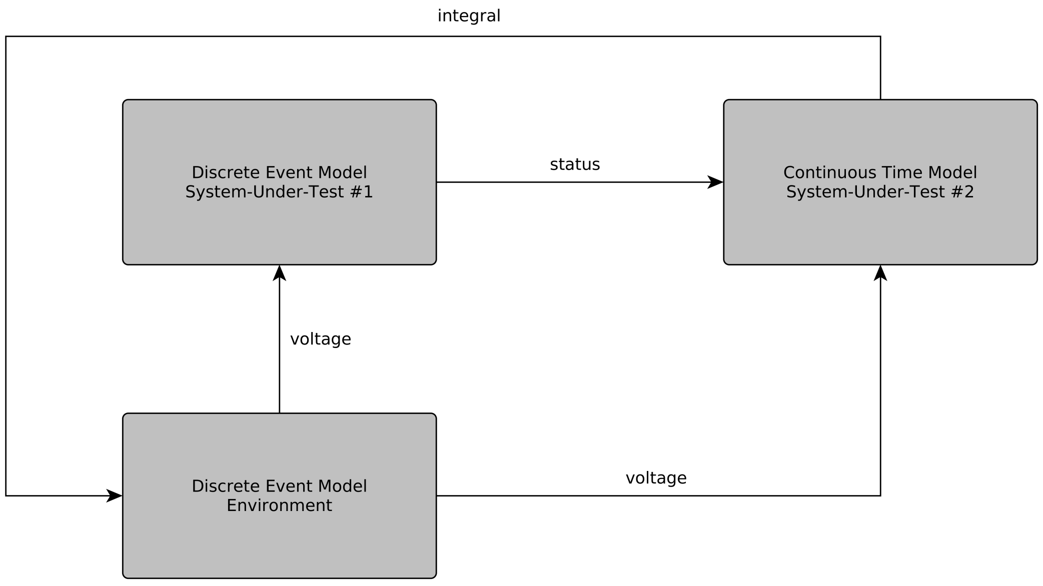

Figure 1: A heterogeneous co-simulation environment which is defined using three models, two discrete-event models and one continuous-time models.

1.2 Requirements

This deliverable is based on and aims at implementing the following high level requirements from [LPH+15].

0005 and 0035 Model checking shall be applied to a co-simulation configuration, rather than stand-alone state charts, and hence the outputs of this requirement flow into the model checking functionality of RTT-MBT. In particular, this requires that the abstracted CT models need to be integrated into the co-simulation environment as well so as to replace the original behavior.

0032 DE models, which are part of a configured co-simulation environment, shall be supported using BMC techniques.

0033 and 0034 This requirement is the key scope of this deliverable, as it specifies that RTT-MBT must support DE abstractions of CT models. Further, DE abstractions of CT models shall be handled as if they originally were DE models. To do so, the DE abstractions should be specified in the same formalism as originary DE models, and can thus be supported by RTT-MBT with as little integration effort as possible.

1.3 Outline

The remainder of this document is then structured as follows. First, Sect. 2 discusses related techniques that have been described in the literature. This section is followed by a survey of the state-of-the-art in DE model checking and a description of the model checking techniques implemented in the RT-Tester system. Directions for different directions of model checking in the INTO-CPS project are then discussed in Sect. 4. Finally, Sect. 5 discusses the configuration items for model checking in INTO-CPS.

2 Related Work

2.1 Bounded Model Checking

The key idea of BMC is to exercise the behavior of a system only up to a certain depth of computations [CBRZ01, CKOS05]. BMC thus merely serves as a bug-hunting framework, as opposed to the approach followed by traditional unbounded model checking techniques [BK08, BCM+90, CES86, CGP99, McM92, McM02], which aims at a formal verification of the analyzed system. However, if the exercised computation depth is just large enough — which is the case if the unrolling depth k of the transition relation has reached the completeness threshold [KOS+11] of the system — then BMC can too be used to prove system properties in the sense that it allows to show the absence of property violations.

2.2 Abstraction

Correct abstractions have been formalized by Cousot and Cousot [CC77] in the abstract interpretation framework. In principle, the semantics of a program is then specified using lattices. Concrete and abstract domains A and C are then connected using an abstraction function $\alpha : A \to C$ and a concretization functions $\beta : C \to A$. For $c \in C$ and a correctly constructed abstraction function α, $\alpha(c)$ then describes c in the sense that it contains c, and possibly more results, which entails soundness. Despite recent progress on automatic generation of abstractions [RSY04, BK10, BK11, BK12], however, designing α typically is still a manual process. Importantly, though possible in theory [RSY04], to the best of our knowledge no research has been contributed to the automatic abstraction of CT semantics into DE semantics, which is the core problem discussed in this document.

However, of course abstract interpretation techniques have widely been applied to the verification of hybrid systems [Hen96]. For example, Sankaranarayanan et al. [SDI08] have combined symbolic model checking with states encoded on top of template polyhedra, that is, conjunctions of linear inequalities $\sum_{i=0}^{n} c_i \cdot v_i \leq k$ where the c_i are fixed a priori. However, such works target an entirely different setting than our work since it is entirely based on abstracting formally specified hybrid automata, whereas we focus on continuous-time models that may not necessarily have a formal semantics (the outputs may, for example, be computed using a controller that is directly connected to the system). Further, the scalability of complex abstractions such as template polyhedra in a network of components is uncertain. As stated by Sankaranarayanan et al. [SDI08, Sect. 1], "*hybrid systems verification is a challenge even for small systems*", which of course applies to networks of hybrid systems.

3 State-of-the-Art in Discrete-Event Model Checking

3.1 RT-Tester Model-Checker

At its core, the model checker integrated into RTT-MBT is based on the well-known bounded model checking (BMC) algorithm for LTL described by Biere et al. [BHJ+06]. The key idea of BMC is to unroll the transition relation of the model step by step, up to a fixed bound k, and check whether the desired LTL specification holds for up to k transitions. Thus, BMC is merely a framework that supports bug-detection, rather than proving the absence of bugs[1]. Typically, SAT or SMT solving is used to implement the BMC algorithm as follows:

- Let $\mathcal{I}$ denote a formula that encodes the initial states of the system.
- Let $\mathcal{T}_{i,i+1}$ denote a formula that encodes a transition from step i to step $i+1$.

[1]It has been shown, however, that BMC can too be used to show the absence of bugs if k is an adequately large value, see Sect. 2.1. Hence, if k is picked adequately as an input parameter to RTT-MBT, the model checking framework in RTT-MBT can be used to show the absence of defects.

- Let φ denote an encoding of the LTL property that is to be checked.
- Then, the LTL specification φ is violated if and only iff the conjoined formula $\mathcal{I} \wedge \bigwedge_{i=0}^{k-1} \mathcal{T}_{i,i+1} \wedge \neg\varphi$ is satisfiable. If the formula is unsatisfiable, the k-times unrolled transition relation does not contain a trace that violates the LTL specification φ.

In RTT-MBT, the above formulae are encoded in a bit-vector theory [KS08] and a dedicated SMT solver called Sonolar is used to check satisfiability.

Encoding transition relations for state charts and encoding these unrollings propositionally so that they can be fed into an SMT solver is a well-known technique [PVL11]. At this point, it is important to note that state charts are themselves inherently concurrent, with the notion of time integrated as a monotonically increasing integral value (as opposed to dense time used in other approaches). This observation suggests that the same techniques that are used to model check a single model, which contains a number of concurrent components, can be used to check a collection of models using the same approach. This issue will be discussed further in the following section.

4 Design Principles for Model Checking Support in INTO-CPS

A key problem for model checking of heterogenous cyber-physical systems specifications is the use of different modelling formalisms to specify the different components. For example, one component may compute its outputs using continuous differential equations, whereas the other one is implemented by means of a state chart. An important decision in the development of the INTO-CPS project was to base model checking purely on DE formalisms, and to over-approximate the behavior of CT components using suitable DE abstractions (cp. Sect. 2.2). The model that is then verified is a network of timed state charts, and abstractions are integrated by replacing CT components with appropriate abstract DE state charts.

4.1 Definition of Model Approximations from CT to DE Models

4.1.1 Interval Abstraction

An classically used abstract domain for program analysis is the so-called interval or box domain, which was introduced for this purpose by Cousot and Cousot [CC77]. In this abstraction, the concrete value of each variable v is described by an interval $[v_l, v_u]$ such that $v \in [v_l, v_u]$ for all possible concrete values of v. An example of an interval abstraction for a continuous variable that evolves over time is given in Fig. 2. The interval domain is probably the coarsest general-purpose abstract domain that has proven valuable in practical applications.

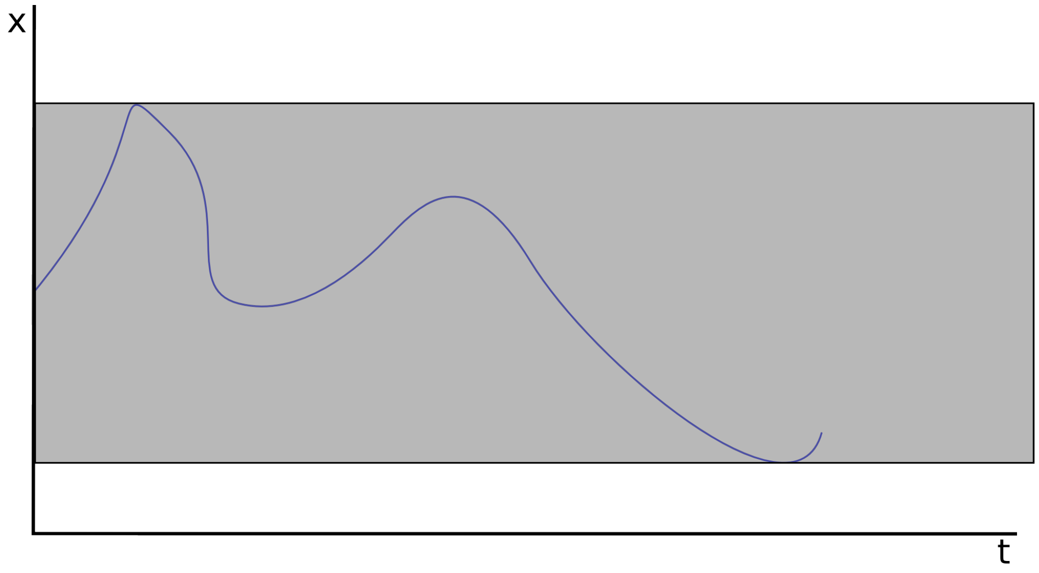

Figure 2: Interval abstraction for a continuous variable.

The FMI 2.0 standard [Blo14, Sect. 2.2.3] already specifies upper and lower bounds for variables using fields called `min` and `max`, which serve as interval bounds. An interval abstraction of a CT variable v can thus simply be synthesized by generating a state chart that consists of a single re-eentrant state directly from the FMU interface specification. The entry action of this state is to assign v a random value in $[v_l, v_u]$ after at least a certain amount of time has passed so as to model discretization of time.

However, it is uncertain how useful the coarse interval abstraction is in practice. During the project, it will be evaluated in how far refinements of this abstraction are required in order to obtain meaningful results. The following section discusses on such refinement.

4.1.2 Gradient-Based Interval Abstraction

The interval abstraction discussed before is coarse in the sense that it does not specify how signal values may evolve over time, as the interval abstraction describes a time-independent invariant that holds unconditionally. However, often the values of continuous variables only change gradually, for example, by at most 5 units within 10 ms. Such information can be integrated into the abstraction, which we discuss by means of an example.

The key idea of the gradient-based interval abstraction is to assign to a signal v values that are invariant for the discretized timeframe (here, 10 ms), which is achieved as follows (see Fig. 3):

- An initial state s_0 is introduced, which models the assignment of the initial value to v. The state contains a single entry action, which is a non-deterministic assignment of values in this range to `v`. For the example, assume that the initial range of `v` is $[0, 20]$, the state thus contains an action v `= rand(0,20)`.
- A second state s_1 is added that models how the value of `v` is updated. In the example, we assume that `v` may change by at most 5 units within a timeframe of 10 ms. To achieve this behavior, the state s_1 is re-entrant after 10 ms and re-assigns a value to v.

Intuitively, the gradient describes the range in which a continuous signal may change in a certain timeframe. An abstract transition relation can thus be derived by simply implementing that a described signal is within the range defined through 1) its value on input to the time frame and 2) the maximally reachable offset during the time frame. The gradient-based interval abstraction can thus be seen as a form of predicate abstraction [GS97], with the predicates derived using just two parameters (the gradient and the timeframe). There are two drawbacks with this approach:

- The unrolling bound k will have to be very large due to the explicit time-discretization. This may lead to intractable models.
- It is difficult to extract the gradients from the CT models automatically. In this case, human intervention will be required to annotate continuous signals with gradients.

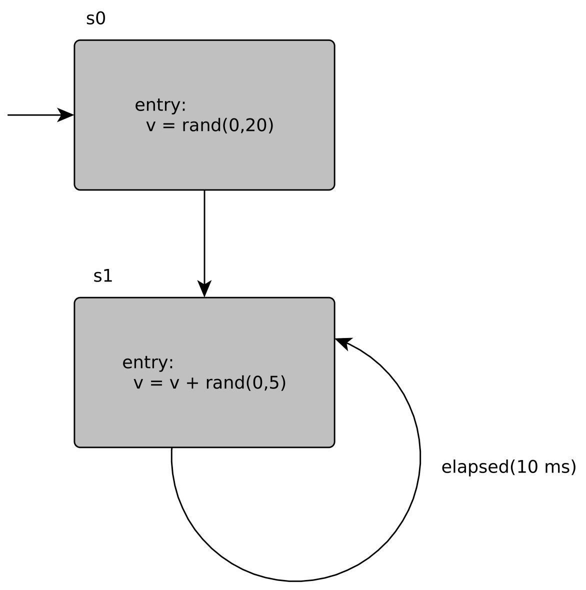

Figure 3: State chart that implements a gradient-based interval abstraction.

4.1.3 Simulation-Based Abstraction

The previous sections have discussed techniques which soundly over-approximate the semantics of CT models using DE abstractions. These approaches may, when applied to industrial-scale multi-model co-simulation environments, lead to some frustration since the abstractions may be either too coarse (cp. Sect. 4.1.1) or too expensive (cp. Sect. 4.1.2). It may thus be advisable to give up on completeness[2] and abstract only parts of the behavior of the CT models.

We suggest that concrete continuous-time traces of CT models could be extracted from executions of the entire co-simulation[3]. Interval-based abstraction techniques could then be applied to these concrete executions. Clearly, this approach is unsound as it does not allow to guarantee the absence of defects in the multi-model co-simulation environment. Yet, it allows to ver-

[2]Unfortunately, it is not possible to quantify the loss of completeness since the simulations only consist of one specific trace through a system. However, combining verification techniques with testing- or simulation-based techniques is a well-known approach [Kur08] if scalability problems or a loss of precision occur, see for example [GK10] for a technique from the context of static program analysis for software.

[3]Of course, the traces extracted are already discretized. However, it is very likely that these discretized traces are so detailed that they are intractable for bounded model checking. We thus plan to further abstract these already discretized traces.

ify properties of the DE models subject to the condition that the CT parts behave similar to a concrete execution of the entire system.

To illustrate, consider Fig. 4, and assume that the signal x indicated by the blue line has been obtained using simulation of a CT model. The signal is identical to the one given in Fig. 2. For this example, we have decided to abstract the concrete signal value with an adaptive interval abstraction with three important properties:

- The signal x is abstracted by a finite sequence of $n + 1$ intervals which we denote $([x_{l,0}, x_{u,0}], [x_{l,1}, x_{u,1}], \ldots, [x_{l,n}, x_{u,n}])$.
- The size of each box is bound from above by a constant c, that is, $x_{u,i} - x_{l,i} \leq c$ for all $0 \leq i \leq n$.
- The timeframe covered by an interval $[x_{l,i}, x_{u,i}]$ is variable. For a continuous-time signal given through a function $f_{\mathtt{x}} : \mathbb{R} \to \mathbb{R}$, the time-frame of validity $[t_{l,i}, t_{u,i}]$ has to be chosen so that $x_{l,i} \leq f_{\mathtt{x}}(t) \leq x_{u,i}$ for all $t \in [t_{l,i}, t_{u,i}]$.

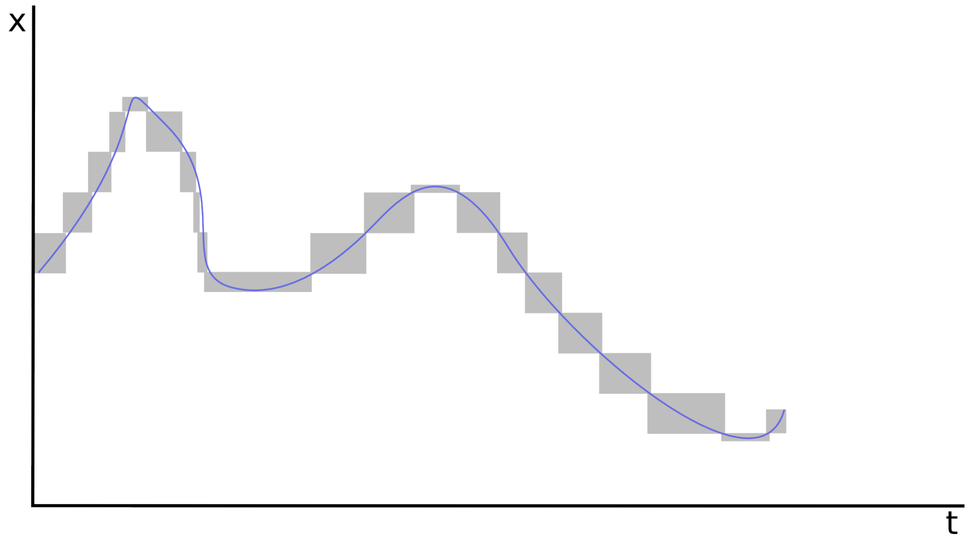

Figure 4: Concrete signal flow from Fig. 2 with adaptive interval abstraction.

Such an abstraction can straightforwardly be represented by a DE model that contains a chain of states. Importantly, the upper bound c on the interval-size provides a means to control the tradeoff between precision and performance of this approach. If c is small, then the abstract model will be precise, but model checking may be intractable. It is then possible to incrementally increase the size of c until model checking terminates in a reasonable amount of time. The impact of c on the number of intervals, and thereby the number of transitions in the abstracted CT model, is highlighted by Fig. 5, where c was chosen twice as large compared to Fig 4.

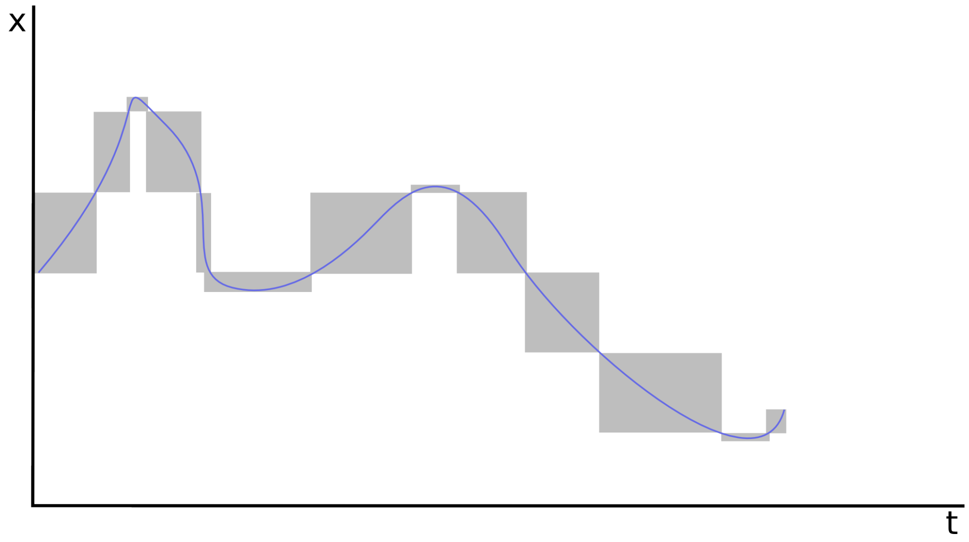

Figure 5: Concrete signal flow from Fig. 2 with adaptive interval abstraction.

Further, it is unclear whether a fixed grid shall be applied for the interval subdivision, or whether it should be adapted. For the examples in Fig. 4 and Fig. 5, respectively, the intervals were defined based on a predefined grid. Fig. 6 shows the results using the same interval size as in Fig. 5, but based on an adaptive grid for which the boundaries of the interval are computed from the concrete signal flow.

5 Configuring Model Checking in INTO-CPS

It is our express aim to depend on as few manual adaptations of the co-model (as expressed in Modelio) as possible to be able to apply model checking. The main configuration artefact for the model checking component in INTO-CPS is thus the co-model. The INTO-CPS application [BLL$^+$15] shall provide further functionality to configure the following parameters:

- Which blocks contained in the co-model shall be abstracted?
- Which abstraction technique shall be applied to which block? Depending on the abstraction technique, additional parameters may become necessary, as sketched in the previous section.

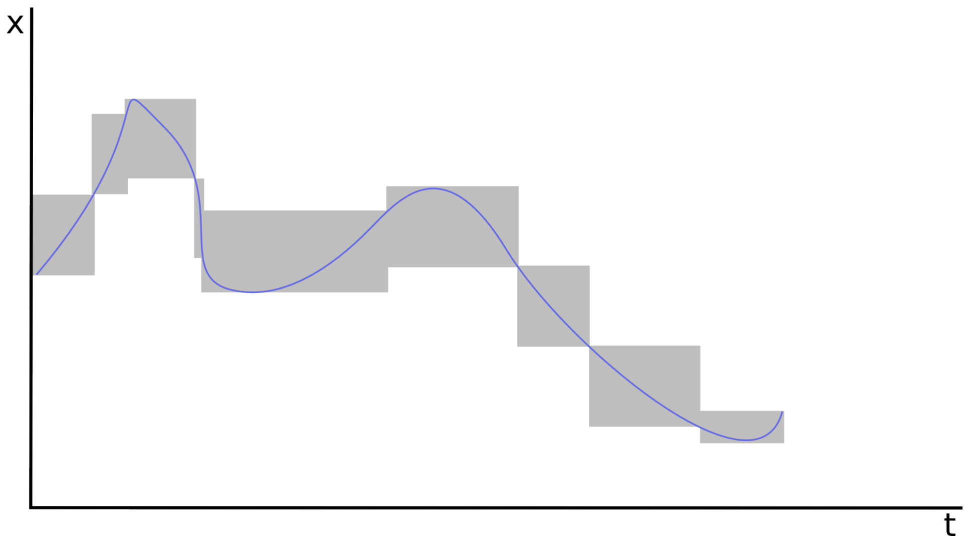

Figure 6: Dynamic interval subdivision scheme.

- Which LTL specifications shall be checked?

Further, it has to be decided at which location in a given FMU the underlying DE model has to be placed (if available). The model checker shall automatically extract this DE model from the respective FMU.

6 Conclusion

This deliverable discusses the overall application setting of model checking in the INTO-CPS project as well as possible abstraction techniques that could be used to integrate DE and CT models. In year 2, these techniques will be implemented and applied to case studies to evaluate their practicability to the real-world problems imposed by the INTO-CPS setting. It is planned that the model checking functionality will be released within the project in M18. Depending on the outcomes of this evaluation phased, it may be necessary to further refine the proposed abstractions using more complex domains.

One particular source of information that has been neglected in this document is the LTL specification to be checked: Interpreting the atomic propositions used in the specification may turn out useful during the generation of the transition relations. For example, the atomic propositions could be used as in predicate abstraction to infer a candidate for an appropriate transition relation. We expect that applying the different model checking techniques to the industrial case studios within the INTO-CPS project will provide sufficient data to evaluate whether such an approach would be valuable.

References

[BCM$^+$90] J. Burch, E. Clarke, K. L. Mc. Millan, D. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *5th Annual Symposiumk on Logic in Comp. Science*, pages 428–439. IEEE, July 1990.

[BHJ$^+$06] Armin Biere, Keijo Heljanko, Tommi A. Juntilla, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), 2006.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[BK10] Jörg Brauer and Andy King. Automatic abstraction for intervals using boolean formulae. In *17th International Static Analysis Symposium (SAS 2010)*, volume 6337 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2010.

[BK11] Jörg Brauer and Andy King. Transfer function synthesis without quantifier elimination. In *Programming Languages and Systems - 20th European Symposium on Programming (ESOP 2011)*, volume 6602 of *Lecture Notes in Computer Science*, pages 97–115. Springer, 2011.

[BK12] Jörg Brauer and Andy King. Transfer function synthesis without quantifier elimination. *Logical Methods in Computer Science*, 8(3), 2012.

[BLL$^+$15] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Sune Wolff, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Marcel Groothuis, and Christian Kleijn. User Manual for the INTO-CPS Tool Chain. Technical report, INTO-CPS Deliverable, D4.1a, December 2015.

[Blo14] Torsten Blochwitz. Functional mock-up interface for model exchange and co-simulation. https://www.fmi-standard.org/downloads, July 2014. Torsten Blochwitz Editor.

[CBRZ01] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming (POPL 1977)*, pages 238–252. ACM, 1977.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[CKOS05] Edmund M. Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Computational challenges in bounded model checking. *STTT*, 7(2):174–183, 2005.

[GK10] P. Godefroid and J. Kinder. Proving memory safety of floating-point computations by combining static and dynamic program analysis. In *19th International Symposium on Software Testing and Analysis (ISSTA 2010)*, pages 1–12. ACM, 2010.

[GS97] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *9th International Conference on Computer Aided Verification (CAV 1997)*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[Hen96] Thomas Henzinger. The theory of hybrid automata. In R.P. Kurshan M.K. Inan, editor, *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[KOS$^+$11] Daniel Kroening, Joël Ouaknine, Ofer Strichman, Thomas Wahl, and James Worrell. Linear Completeness Thresholds for Bounded Model Checking. In *23rd International Conference on Computer Aided Verification (CAV 2011)*, volume 6806 of *Lecture Notes in Computer Science*, pages 557–572. Springer, 2011.

[KS08] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[Kur08] R. P. Kurshan. Verification technology transfer. In *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 46–64. Springer, 2008.

[LPH+15] Peter Gorm Larsen, Ken Pierce, Francois Hantry, Joey W. Coleman, Sune Wolff, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagić, Jörg Brauer, Etienne Brosse, Carl Gamble, and Simon Foster. Requirements Report year 1. Technical report, INTO-CPS Deliverable, D7.3, December 2015.

[McM92] Kenneth L McMillan. *Symbolic Model Checking.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1992. Kluwer Academic Publishers, ISBN: 0-7923-9380-5.

[McM02] Kenneth L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 250–264. Springer, 2002.

[PVL11] Jan Peleska, Elena Vorobev, and Florian Lapschies. Automated Test Case Generation with SMT-Solving and Abstract Interpretation. In Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *Nasa Formal Methods, Third International Symposium, NFM 2011*, pages 298–312, Pasadena, CA, USA, April 2011. NASA, Springer LNCS 6617.

[RSY04] Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. Symbolic implementation of the best transformer. In *5th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2004.

[SDI08] S. Sankaranarayanan, T. Dang, and F. Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2008.

A List of Acronyms

BMC	Bounded Model Checking
COE	Co-simulation Orchestration Engine
CPS	Cyber-Physical Systems
CT	Continuous-Time
DE	Discrete Event
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
LTL	Linear Timed Logic
MBT	Model Based Testing
RTT	RT-Tester
RTT-MBT	RT-Tester Model-Based Test Case Generator