Model-Driven Methods to Design of Reliable Multiagent Cyber-Physical Systems* **

Sergey Staroletov[1], Nikolay Shilov[2],
Vladimir Zyubin[3], Tatiana Liakh[3], Andrei Rozov[3],
Ivan Konyukhov[2], Innokenty Shilov[4],
Thomas Baar[5], Horst Schulte[5]

[1] Polzunov Altai State Technical University, Barnaul, Russia
[2] Innopolis University, Russia
[3] Institute of Automation and Electrometry SB RAS/
Novosibirsk State University, Novosibirsk, Russia
[4] Gromov Flight Research Institute (GFRI), Moscow, Russia
[5] University of Applied Sciences Berlin (HTW Berlin), School of Engineering

Abstract. Cyber-Physical Systems (CPS) are real-world systems that use a cyber part to control a physical part; hybrid systems are virtual-world systems to model CPS. In this paper we address several problems related to CPS-design and argue advantages of a Model-Driven Developing (MDD) approach to CPS-design. We study a simple car stopping system and show that such systems can be modeled without any code writing. However, safety isn't clear even for this simple system and testing/simulation isn't sufficient to prove safety, but CPS MDD should be supported by a formal verification also. We examine modeling, simulation and verification tools and show how our approach can be applied. We also introduce a concept of cyber-physical Believe-Desire-Intention (BDI) agent and demonstrate how cooperative agents of this type can predict locations of partners.

Keywords: Cyber-Physical Systems · Model-Driven Developing · Believe-Desire-Intention agents.

1 Introduction

We understand cyber-physical systems (CPS) as real-world systems that use a cyber part (computer/software) to control or operate a physical part (hardware/physical process). We define hybrid systems as a virtual-world systems that combine continuous real functions with discreet state machines, they are models of the cyber-physical systems. The presence/use of the continuous real functions makes unsuitable techniques which are based on pure discrete state-transition systems and integer arithmetic.

Hybrid Systems may be represented by Hybrid Programs, and specified using the Hybrid Dynamic Logic [1, 2]. For example, the syntax of hybrid programs in [3] is

* Partially supported by DFG program Initiation of International Collaboration

defined as follows:

$$\alpha ::= x := e \mid ?Q \mid x' = f(x)\&Q \mid \alpha \cup \alpha \mid \alpha;\alpha \mid \alpha^* \tag{1}$$

where α is a meta-variable for the the hybrid programs, x is a meta-variable for program variables, e is a meta-variable for the first-order terms on real numbers, f is a meta-variable for the continuous real functions, and Q is a meta-variable for the first-order formulas over real numbers. The construct ";" means here the sequential composition, "$\cup$" — is the non-deterministic choice, "?" — is the test operator, and "*" — is the non-deterministic iteration (like Kleene-star). Hybrid Program is a representation of Hybrid Automaton [4].

Modelling and computer implementation of CPSs is not a trivial one-step process, CPSs should be accurately modeled as a hybrid system by teams of physicists, mathematicians problem-domain (e.g. civil, mechanical, electric, chemical etc.) and control engineers.

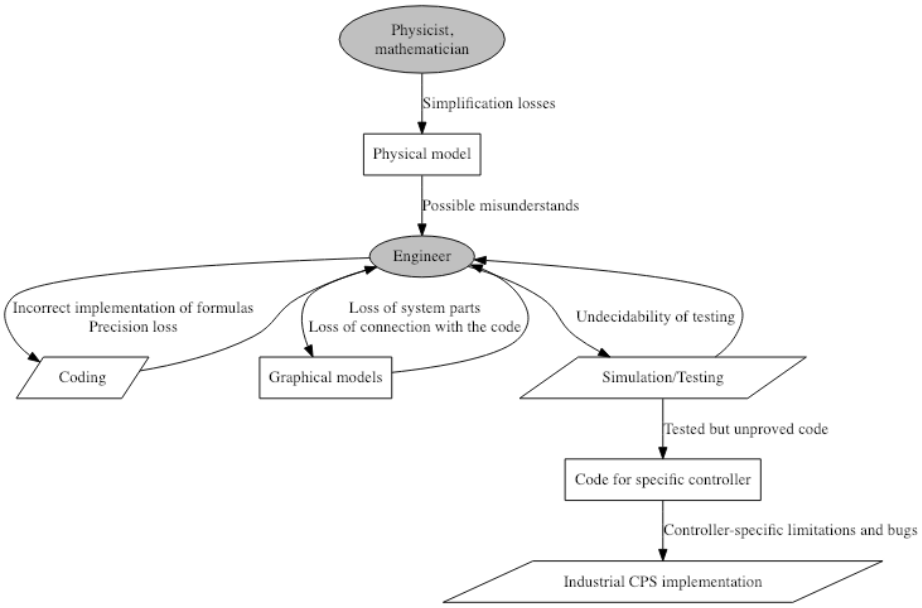

Fig. 1. Possible problems during implementation of the CPSs

Presently there are several ways to implement on computer the physical part of CPSs:

- as programs that change the variables according to in-coded exact solutions of the differential equations; the exact solutions can be found analytically or by using symbolic computer algebra tools like Mathematica, Reduce or Maxima;
- as programs that numerically solve the differential equations in run-time using, for example, the Runge-Kutta or Euler methods (in simple cases) [5, 6].

The Model-Driven approach (MDD, stands for Model-Driven developing) [7] may be an alternative to the above. It is based on executable diagrams (schemes) for which program code can be generated from them automatically. In the paper, in particular, we address the safety of CPSs created with the MDD approach.

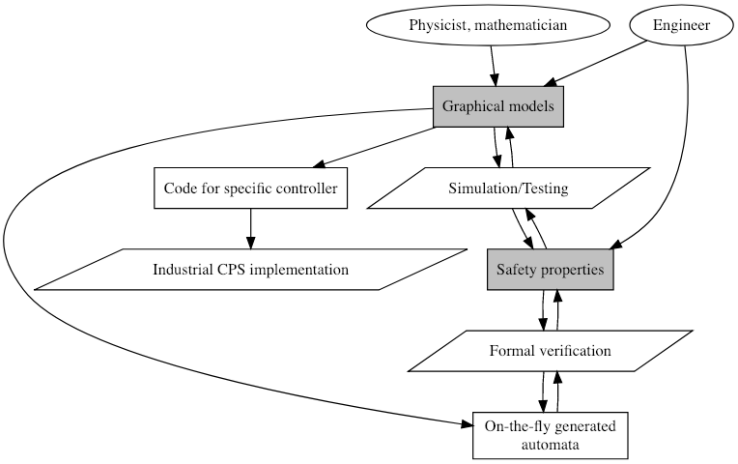

Fig. 2. MDD of CPS process with verification of safety properties

A possible work-flow of CPSs implementation in the industrial context is shown on Fig. 1. On this diagram we enumerate some problems that may be caused by lose of some details due to change of the abstraction level.

The process begins with the creation of mathematical or physical models by engineers, who have experience in the specific problem domain and are able to use appropriate methods to describe the system in models. As a result, the models have been created, and here we can see possible losses (the model describes the process with some simplifications).

Then the model of the process comes to engineer who is in charge for implementation on a device (e.g. controller). The engineer can have misunderstands of the model and therefore, implement the wrong model. Implementing the system, the engineer creates code in one of the programming languages to solve the equations (Physical part) and control features (Cyber part of the CPS). Here the engineer can make mistakes also in the implementation of the mathematical formulas because of the use of floating point formats instead of mathematical real numbers. Also loss of some system behavior, code and model mismatching, etc. can happen in the implementation stage. During a CPS construction, the engineer periodically makes simulations of the created code by running it with some pre-defined test inputs and observing resulting outputs, for example, looking at a plot with the system values evaluation. We note that such simulation is a testing process, it can detect errors and mistakes but not to prove that the implementation is error-free [8]. Moreover, even if the engineer believes that the code acts as

correct CPS (because all tests have passed), we need some guaranties whether the code actually represents the CPS initially been expected.

As a result of the current process, tested but unproved code goes into a controller (which in addition, has own bugs and limitations) to operate with the real system.

2 Model-Driven Developing of Cyber-Physical Systems

MDD is a way to create a software system from scratch by drawing some diagrams or writing textual models and then simulate or execute these diagrams and generate the system code from them. Taking into account the problems discussed in the section 1, we can distinguish the following main advantages of applying MDD in the CPSs design process:

- engineers can see a graphical model or a DSL [9] code and move through it;
- no code and model distinctions after changing any one of both;
- possible ways to generate code for different target languages and purposes (for example, for Matlab Simulink or for implementation in a particular controller);
- the system can be created by engineers who know how to describe physical processes but do not have proper programming skills.

So, MDD may be considered as a realistic way to organize work of large teams of engineers to implement CPSs and to improve the quality. Consequently we can update the process shown in Fig. 1, and move to the MDD of CPS design process depicted in Fig.2.

According to the process, physicists, mathematicians and engineers work together with a CPS represented by graphical models. They can create symbolic nodes which model different parts of the system and determine relations between them. The engineers also detail the requirements expressed in terms of timed logic (see [10] for example). (Remark, that we are mostly interesting in the safety properties [11], express that *"something (bad) will never happen"* during a system execution.)

As soon as graphical models and requirements are constructed, it is possible to simulate the system behavior without any code writing. Having safety properties, it is possible to add appropriate assertions (contracts) that may be checked in run-time. Then it may make sense to create on the fly different types of Hybrid Automata which models the Hybrid Program and try different verification tools when needed. And of course, code for a real controller can be generated from the model automatically.

3 Related works

3.1 CPSs modeling languages and tools

Certainly, it is possible to implement the Physical part of a CPS by writing code in a programming language (like C or Java). Dealing with a sophisticated system, it is required to use additional programming libraries to solve different types of ODEs. To realize the Cyber part, network or special hardware libraries should be used. To implement agent-based interoperations, frameworks and languages conform to actor approach [12] can

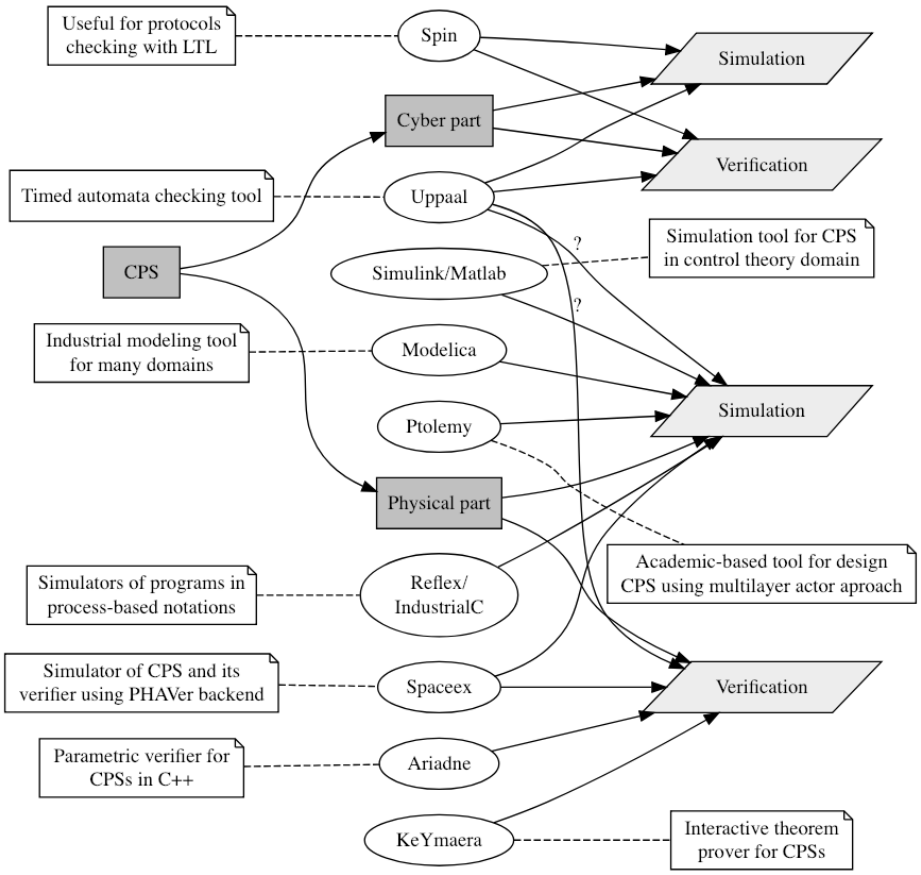

Fig. 3. A survey of CPSs simulation and verification tools

be used, for example, Erlang language or Akka library for Java/Scala. Such systems can be organized as multilayer applications with parts implemented in different languages and even act as microservices.

A process-oriented technology has been introduced in [13] to express control software as a set of interacting processes, which are extended automata with special operators that implement concurrent flow control and time-interval managing. This approach has been implemented in a family of languages such as Reflex [13] and IndustrialC [14]. With native support for state machines and floating point operations, these languages allow Hybrid Systems to be easily expressed in code.

In the engineering world, currently, a lot of engineers use Matlab Simulink©. It offers to create the CPSs by using a graphical representation very close to the control theory notations. But it lacks of control automata creation in the graphical way, and some portions of the system are still developed by writing lines of code in Matlab language [15].

Another industrial modeling tool is Modelica© [16]. It is a propriatory software that allows to model the dynamic behavior of CPS using different components for mechanical, electrical, thermal, hydraulic, pneumatic, fluid, control and other domains. Models can be described by differential, algebraic, and discrete equations. A special model input language is used, but there are also graphical editors for it.

Ptolemy by Berkeley [17] is a free academic-based tool that supports construction of CPS from small actors at different layers of abstraction. It supports system presentation as a multilayer agent that communicate with input/output ports; each agent can represented as a combination of actors and other agents [18]. Actors could be plotting tools, mathematical operations (e.g. differentials), etc. Layers can be automata or state charts, and each layer is controlled by a controller, which can be discrete or continues time unit. The Ptolemy is very extensible, so a developer can implement own actors based on pre-defined interfaces and export the whole system or an actor as Java classes.

3.2 CPSs verification tools

Fig. 3 presents some tools for simulation and verification of CPSs. Presented tools may be classified into two categories:

1. tools which provide support both — CPSs modeling and simulation;
2. tools which support only CPSs verification (checking some properties).

Uppaal [19] is a model checker and simulator for automata-based programs. User can create a model of a system, including various components in the form of an extended timed automaton and then query some properties (using a variant of LTL formulas). User can provide invariants to the model states which can contain timed based derivatives and check them at run-time. Uppaal has some problems with floating point variables and values.

Spin [20] is a model checker of programs in an actor-based modeling language Promela (stands for *Protocol meta-language*) with respect to given LTL formulas. It is a powerful tool to prove the interprocess communications and protocols. It lacks floating point support too; therefore, it cannot be used for full CPSs verification, it can be helpful only to check the Cyber part of the systems with interoperations.

SpaceEx tool [21] is distributed as a virtual machine with web-server and includes a model editor, a simulator of CPSs and a converter to PHAVer [22] to verify the properties of such systems over infinite time. It also has model converters from other systems, for example, Simulink, to transform their notation to SpaceEx program.

Ariadne [23] can be useful to create non-linear CPSs in C++ code, simulate and verify them. It offers a parametric verification, which provides intervals for values of system variables to preserve the given requirements of the system. Therefore it is possible to use Ariadne with some code checking methods to provide verification of the whole CPS.

KeYmaera [24] is an interactive theorem prover based on Dynamic Logic for the CPSs expressed as Hybrid Programs (HPs). The user can prove properties of the system by providing some simplification tactics, or it can be done automatically, that is why the range of systems to be proved in KeYmaera is potentially very wide. The tool can connect to some mathematical systems or solvers at each step of simplification.

Please refer our paper [25] for a more detailed survey of these CPSs modeling and verification tools. Please refer also the cited paper for example of verification of a safety property of a simple SPC in C code with Frama-C [26] WP tool which is based on the contracts approach.

4 A case study: MDD approach to car braking system

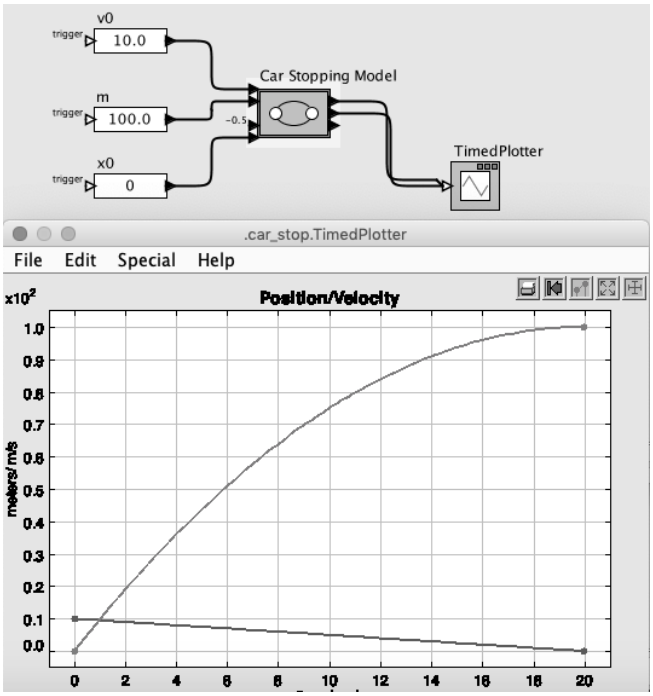

Fig. 4. A top level CPS model and a simulation result

Let us consider a car moving with a deceleration according to the following equation:

$$\begin{cases} x' = v \\ v' = a \\ a = -\dfrac{v_0^2}{2 \cdot (m - x_0)} \end{cases} \tag{2}$$

The negative acceleration a is calculated based on observation at the time t_0 (with the speed v_0 and initial position x_0) to a given obstacle m according to a school physical law *"moving with a constant acceleration"*.

In this section we model the example system with Ptolemy tool to demonstrate how to use the MDD approach to design and simulate such very simple CPS.

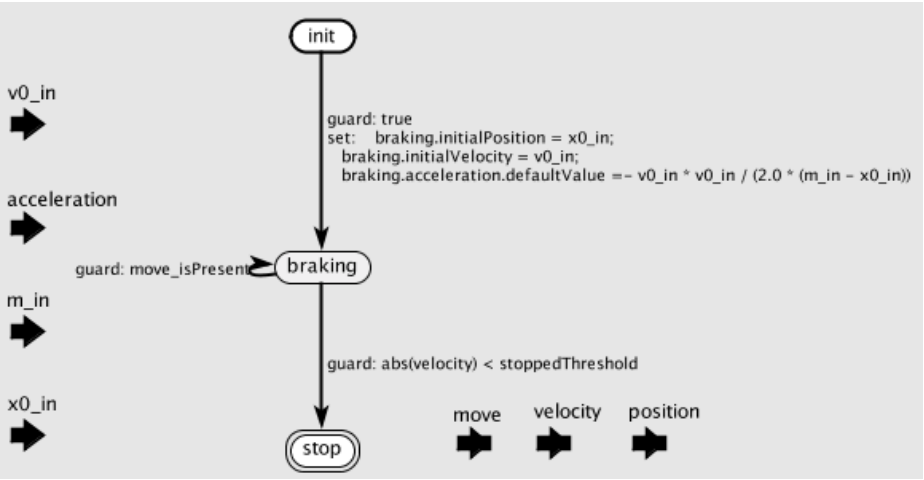

Fig. 5. The CPS model inside

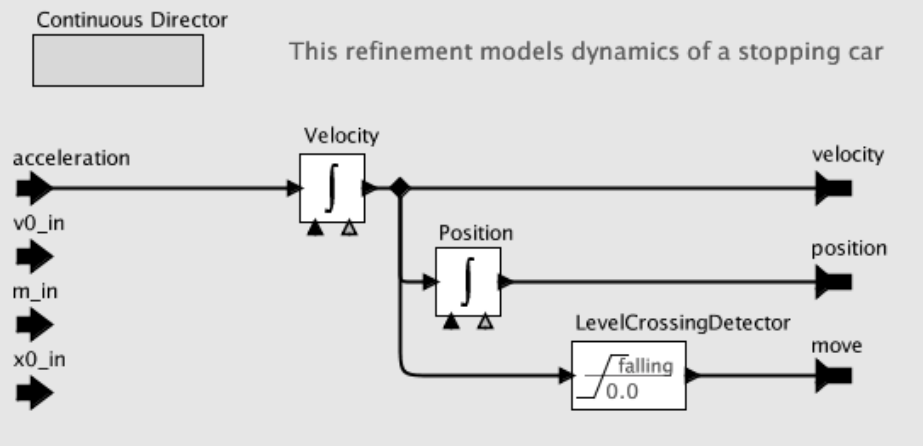

Fig. 6. Braking state refinement

In Fig. 4 the top layer model and modeling results are shown. It just plots two model parameters — current position and slowdown of current velocity.

In Fig. 5 the model is shown from "inside" as an automaton with three states – *init*, *braking* and *stop*, where *braking* – is a Hybrid state that is the subject to further decomposition.

In Fig. 6 the braking physical model is actually implemented. It means that the first derivative of the position x is the velocity v, the second derivative of the position is the acceleration a. So, the system model is very close to the real process description, and that simulation doesn't require any program code writing.

5 Verification problem

For the sample system presented in the previous section 4, the safety property could be formulated as *"the car position should be always less the position of the obstacle"*. It can be expressed as LTL formula (3):

$$G(x < m) \tag{3}$$

One who looks to the graph in Fig. 4 could say that for the system variables this safety property holds, and the simulation proves it. But as we derived before, the simulation is only a process of quality assurance very similar to testing. For example, the graph can miss sudden function changes due to the small grid size, so for reliable safety systems, we need a mathematically sound way to prove the system properties. That way is the formal model verification process [8].

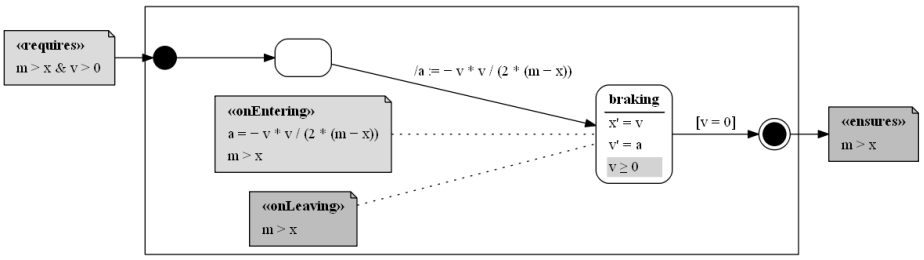

Fig. 7. Control-flow graph with preconditions, postconditions and invariants [25] as a model of the simple CPS (2)

A way to help engineers to start using formal verification — is to integrate the verification process into the tool which is in use for design CPS with MDD approach. In this way, the engineers can apply formal verification without actually getting know how the verification works.

As we have mention above in the section 3, Matlab Simulink includes validation instruments. However, these instruments are not of formal verification kind. Currently,

the tool is a closed-source and a closed-architecture program, so it is unknown whether it is possible to plug additional model checkers in.

The Ptolemy tool has included a model converter [27] to use with some model checkers [28], but now it can convert only discrete time actors. Generation of continues time checking actors, in principle, is possible since the tool is extensible. But the question is: what back-end should be used actually to verify the models?

In the paper [25] a graphical way to specify and verify CPSs has been proposed. It can help to prove safety properties by introducing some contracts in control states. For the simple model, a generated control graph is presented in Fig. 7, so after generating proof obligations and HP in KeYmaera theorem prover input language it is possible to prove this system automatically. In mathematical notation, for the system (3) we have the following safe contract for the long-running state *braking*:

$$(a = -v \cdot v/(2 \cdot (m-x)) \wedge (m > x) \rightarrow [\{x' = v; v' = a\}](m > x) \cup (v = 0)).$$

This contract enables proving the safety property automatically without any user interaction in the theorem prover.

6 Towards Multi-agent Cyber-Physical Systems

In this section we proceed to study multi-systems with a number of *cyber-physical BDI-agents*.

A *distributed system* consists of multiple autonomous individual programmable computers that communicate through a network. Communication (in a distributed system) is said to be fair, if every computer which needs to communicate with any other will communicate eventually. (Of course, some communication scheduler or mechanism is required to guaranty the fairness.)

A *multi-agent system* is a distributed system [29, 30] that consists of *agents*. A *BDI-agent* [31] (just *agent* in the sequel) is an *autonomous*, *reactive* and *proactive* object (in OO-sense) whose internal states may be characterized in terms of *Beliefs* (B), *Desires* (D), and *Intentions* (I). Agent's beliefs represent its ideas/opinion about itself and the *environment* that includes other agents and the network; these ideas/opinions may be incorrect, incomplete, and (even) inconsistent. Agent's desires represent its long-term aims, obligations and purposes (that may be controversial). Agents' intentions are used for a short-term planning. Agent's logical omniscience means that an agent immediately knows all logical consequences that follow from its knowledge.

We distinguish belief(s) and knowledge according to Plato, that knowledge is a true belief, i.e. a judgment/statement that has a validation [32–34]. Thus we assume in this paper that a belief of some agent becomes its knowledge in some of its individual states if there exists any formal proof of the belief (i.e. we assume agent's *logic omniscience*).

Reactivity means that an agent can change its beliefs after interaction with other agents and deliberation. Proactivity means that an agent can change its intentions (i.e. to plan its nearest future behavior) after change/update of its beliefs and deliberation.

Every agent is autonomous, i.e. a change of its personal beliefs and intentions can't be decreed by any other agent. A rational agent has clear preferences and always chooses an action (in feasible actions) that leads to the best (individual or group)

outcome. A bounded rationality is decision making limited by the cognitive abilities of agents (e.g. the finite amount of time they have to make decisions).

A multi-agent algorithm is a distributed algorithm (i.e. protocol of distributed system) that solves some problem by means of cooperative work of agents in a multi-agent system. A related paradigm is algorithmic mechanism design [35].

Fault-tolerance of a multi-agent algorithm is (as for distributed algorithms) an ability to solve the problem correctly in spite of (partial) network failure and/or incorrect behavior of some of individual agents. A knowledge-based algorithm paradigm [32] assumes that any modification of any shared resource may/can be attempted by an agent only when the agent knows that the access for modified resource is safe (i.e. is races-free in particular).

Recall that CPS is a system with the Physical (real-world process) part with a continuous behavior (modeled by differential equations) controlled by a programmable computer. A cyber-physical agent is an agent that is CPS itself.

A cyber-physical BDI-agent in a cyber-physical multiagent system

- knows the differential equation(s) that describes its physical behavior, controls all parameters of this (these) equation(s);
- has a belief about equations that model physical behavior of all other agents in the system and ranges of their control parameters.

Let us discuss examples of problems to illustrate the difference between just BDI-agents and cyber-physical BDI-agents. Both problem statements have the same start:

> There are $N > 1$ autonomous agents (”robots”) and the same number of shelters in general position on a plain part of Mars. Locations of all shelters are fixed and known to all robots. Each robot knows a shelter that was assigned to it from the very beginning, its own distances to all shelters.

But then problems differ.

The first problem is *Mars Robot Puzzle* described, solved and (manually) verified in [36] using system of BDI-agents, its statement has the following continuation:

> Each robot doesn't know locations of any other robot. All robots can communicate with each other in P2P-manner and every pair of communicating robots may swap their shelters. All robots have to select individual shelters to move in by a straight route. Definitely, robots should not collide (it means that their routes should not intersect). Hence, every individual robot can move to a shelter only when it knows for sure that it will not collide with any other robot on the route. Problem: Design a multi-agent knowledge-based algorithm that guarantees that every robot will eventually know that its route to the selected shelter does not intersect with routes of other robots.

Let us refer the next problem which we would like to refer *Mars-rovers Planning Problem*. Its statement has the following continuation:

> All robot can communicate with each other in P2P-manner and ask/inform each other about their current locations (if requested). All robots have move to the

initially assigned individual shelters without approaching each other to close (with individual safety distances). Problem: Design a multi-agent knowledge-based algorithm that guarantees that every robot will eventually reach the assigned shelter safely (i.e. always being on safe distance with all other robots).

This problem can be considered as a special case of the motion-planning [37]. Formal specification and verification (including computer-aided verification) of this problem is one of our topics for further research. In the next section we present first step in this direction — formal specification and (manual) verification of a method how a cyber-physical BDI-agent can predict behaviour of others in a cyber-physical multi-agent system.

7 Towards predictive cyber-physical agents

Agent's proactivity assumes deliberation i.e. that an agent can "predict" behavior of other agents in a multi-agent system (i.e. it has a belief about behavior of others). Recall also that a BDI-agent in a cyber-physical multi-agent system

- knows the differential equation(s) that describes its physical behavior, controls all parameters of this (these) equation(s);
- has a belief about equations that model physical behavior of all other agents in the system and ranges of their control parameters.

So it implies that an individual belief of a cyber-physical BDI-agent about any particular other agent (a partner) of the system can be represented by a differential equation

$$F(t, y, y', \ldots, y^{(n)}) = 0$$

in which F is a real parameterized expression where

- parameters of the expression represent partner's control features in which the agent believe,
- the first argument t is current time,
- the second argument $y = y(t)$ is a relative instant location of the partner (with respect to the agent itself) at time t,
- the third argument $y' = y'(t)$ is a relative velocity, i.e. the first derivative of $y = y(t)$,
- etc.

Let us consider in this section the most simple form for the above equation

$$y' = f(t, y)$$

where f is a *known* explicit real expression (i.e. we assume that the agents knows all control features of itself and the partner). Let us discuss how the agent can predict their joint with the partner behavior maintaining some safety condition, i.e. the following prediction problem: the agent

- knows the expression for f that describes relative location of the partner and its initial relative location at time t_{ini} is y_{ini},

– would like to predict the final relative location y_{fin} at time $t_{fin} = t_{ini} + \Delta$ and guarantee that finally they approach each other at distance not less than some known safety radius $R > 0$.

(Remark that we consider here a *weak* safety requirement while one may be interested in a more *strong* safety condition stating that the agent and the partner never approach each other at distance less then R.)

The problem can be solved, in particular, using the well-known Euler method [6] presented below:

– pick-up some $n \geq 1$, $n \in \mathbb{N}$, such that step $h = \frac{\Delta}{n}$ is believed to be sufficient for a desired accuracy and let $t_0 = t_{ini}$ and $u_0 = y_{ini}$;
– for each $m \in [0..(n-1)]$, if t_m is defined already then let $t_{(m+1)} = t_m + h$, $d_m = h \times f(t_m, u_m)$, and $u_{(m+1)} = u_m + d_m$;

The output consists of a tabular function u that maps each t_m $(0 \leq m \leq n)$ to u_m and approximates the exact solution of the equation $y' = f(t,y)$ in these points in general and the value u_n approximating y_{fin} in particular. The flowchart of the algorithm that implements a variant of the method to compute just an approximation for the final value y_{fin} is presented in the Fig. 8.

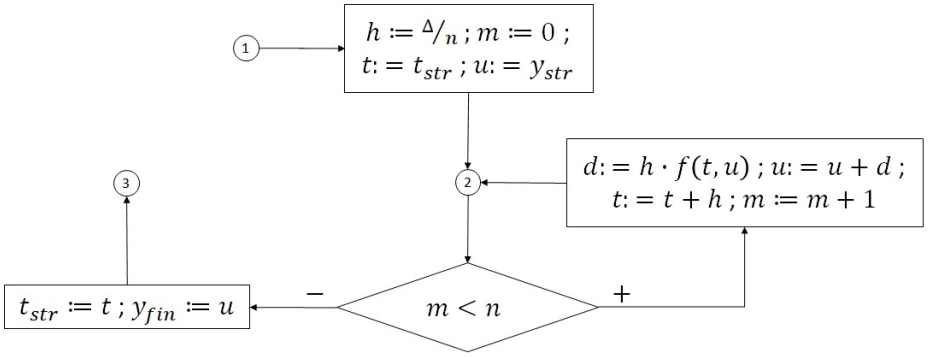

Fig. 8. The flowchart of an *Euler Algorithm* that computes an approximation for the final value using Euler method

A quite common variant of applicability (sufficient) conditions for the Euler method [6] comprises the following three properties:

1. $\Delta > 0$ and $n > 0$ are positive real and integer numbers, $t_{ini} \in \mathbb{R}$ and $t_{fin} = t_{ini} + \Delta$ are reals;
2. $f : [t_{ini}, t_{fin}] \times \mathbb{R} \to \mathbb{R}$ is uniformly Lipschitz continuous in the second argument (formally defined below);
3. there exists a twice continuously differentiable solution $y : [t_{ini}, t_{fin}] \to \mathbb{R}$ of the initial value problem $y' = f(x,y)$, $y(t_{ini}) = y_{ini}$.

Recall that a function $g : \mathbb{R} \to \mathbb{R}$ is Lipschitz continuous, if there exists a real constant $q > 0$ such that $|g(x+\delta) - g(x)| \leq q \cdot \delta$ for all $x \in \mathbb{R}$ and positive $\delta \in \mathbb{R}^+$. In particular, Lipschitz continuity in the second argument for a function $f : [t_{ini}, t_{fin}] \times \mathbb{R} \to \mathbb{R}$ means that for every $t \in [t_{ini}, t_{fin}]$ there exists a real constant $q > 0$ such that $|f(t, x+\delta) - f(t,x)| \leq q \cdot \delta$ for all $x \in \mathbb{R}$ and positive $\delta \in \mathbb{R}^+$. In contrast the above condition 2 about an uniform Lipschitz continuity in the second argument for function $f : [t_{ini}, t_{fin}] \times \mathbb{R} \to \mathbb{R}$ means that there exists a real constant $q > 0$ such that $|f(t, x+\delta) - f(t,x)| \leq q \cdot \delta$ for all for $t \in [t_{ini}, t_{fin}]$, $x \in \mathbb{R}$, and positive $\delta \in \mathbb{R}^+$; let us refer the constant q in this case as *Lipschitz constant* for f in the second argument.

Let $a, b \in \mathbb{R}$, $a \leq b$ and $n \in \mathbb{N}$ be real and natural numbers. A function $g : [a,b] \to \mathbb{R}$ is n-times continuously differentiable (notation $g \in C^{(n)}[a,b]$), if the function has derivatives $g^{(0)} \equiv g : [a,b] \to \mathbb{R}$, ... $g^{(n)} : [a,b] \to \mathbb{R}$, and all these functions are continuous on $[a,b]$: $g^{(0)}, \ldots g^{(n)} \in C[a,b]$. In particular the above condition 3 says that $g \in C^{(2)}[t_{ini}, t_{fin}]$, i.e. that y, y', and y'' are defined and continuous at each point of $[t_{ini}, t_{fin}]$. Recall *the extreme value theorem* (also known as Weierstrass theorem) states that if a real-valued function g is continuous on the closed interval $[a,b]$, then g must attain its maximum and minimum (least once each); hence the condition 3 implies that there exists a real constant $p > 0$ that is an amplitude for $|y''|$ (i.e. such that $|y''| \leq p$ for all $t \in [t_{ini}, t_{fin}]$).

Thus the above variant of applicability conditions for the Euler method for a predictive cyber-physical BDI-agent can be reformulated as follows:

1. $\Delta > 0$ is a time-interval after which the agent would like to know a relative location of the partner, $n > 0$ is a positive integer number that the agent believes to be sufficient for partitioning the time-interval Δ, t_{ini} and $t_{fin} = t_{ini} + \Delta$ are the initial and the final time readings known for the agent;
2. $f : [t_{ini}, t_{fin}] \times \mathbb{R} \to \mathbb{R}$ is a known for the agent law that returns partner's relative speed (as a function of time and relative location) and that the agents knows that the function is uniformly Lipschitz continuous in the second argument with Lipschitz constant $q > 0$;
3. the agent knows the initial relative location of the partner y_{ini} at time t_{ini} and that the relative acceleration of the partner is a continuous function on $[t_{ini}, t_{fin}]$ with some constant amplitude $p > 0$.

Let us denote all these three conditions altogether as *Initial Knowledge*.

The above variant of applicability conditions for the Euler method [6] guarantees that upon termination of the method for all $m \in [0..n]$

$$|y(t_m) - u_m| \leq \frac{p}{2q}\left(e^{q(t_m - t_{ini})} - 1\right)h$$

where q is the Lipschitz constant for f in the second argument and $p > 0$ is an amplitude of y'' on $[t_{ini}, t_{fin}]$. Since we are interested in the prediction of the final relative location exclusively then this property transforms into the following *Guaranteed Location* condition:

$$|y(t_{fin}) - y_{fin}| \leq \frac{p}{2q} \times \frac{\Delta}{n} \times \left(e^{q\Delta} - 1\right).$$

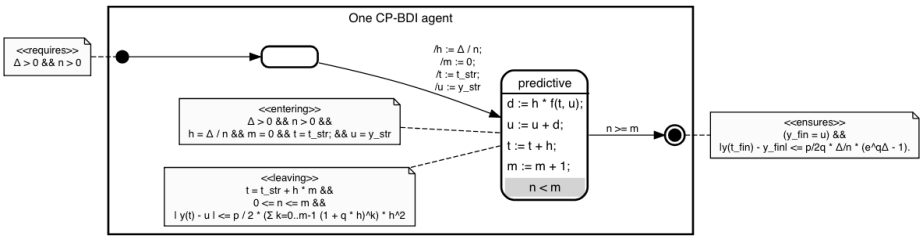

Fig. 9. Control-flow graph with preconditions, postconditions and invariants as a model of the predictive cyber-physical BDI agent

Conditions *Initial Knowledge* and *Guaranteed Location* altogether with the *Euler-Algorithm* give us the following total correctness assertion [38]

$$[\mathit{Initial\ Knowledge}]\ \mathit{Euler\ Algorithm}\ [\mathit{Guaranteed\ Location}]$$

that states that

if the agent has the *Initial Knowledge*
then the *Euler Algorithm* terminates and
the *Guaranteed Location* holds upon termination.

If one would like to prove the assertion formally then

- the precondition *Initial Knowledge* should be assigned to the control point 1, the postcondition *Guaranteed Location* should be assigned to the control point 3 in the flowchart in Fig. 8,
- the invariant consisting of the following two properties
  - $0 \le m \le n$ and $t = t_{ini} + h \cdot m$,
  - $|y(t) - u| \le \frac{p}{2} \times \left(\sum_{k=0}^{k=m-1} (1 + q \cdot h)^k\right) \times h^2$

  should be assigned to the control point 3 in the flowchart in Fig. 8.

The proved total correctness assertion gives us an opportunity to provide an estimation for n that guaranties the weak safety condition. Really, since

$$|y_{fin}| - |y(t_{fin})| \le |y_{fin} - y(t_{fin})| \le \frac{p}{2q} \times \frac{\Delta}{n} \times \left(e^{q\Delta} - 1\right)$$

then

$$|y_{fin}| - \frac{p}{2q} \times \frac{\Delta}{n} \times \left(e^{q\Delta} - 1\right) \le |y(t_{fin}|;$$

so $R < |y(t_{fin})|$ if $\left(R + \frac{p}{2q} \times \frac{\Delta}{n} \times \left(e^{q\Delta} - 1\right)\right) < |y_{fin}|$, i.e. in the case when

$$|y_{fin}| > R \text{ and } n > \frac{p}{2q} \times \frac{\Delta}{|y_{fin}| - R} \times \left(e^{q\Delta} - 1\right).$$

8 The control-based verification of cyber-physical agents

In the previous section 7, we stated that the predictive cyber-physical BDI-agent problem can be reduced to contract verification. In the mathematical way, with the variables introduced before, staying in a long-running state could be described as

$$(\Delta > 0) \wedge (n > 0) \wedge (h = \frac{\Delta}{n}) \wedge (m = 0) \wedge (t = t_{str})$$

$$\wedge (u = y_{str}) \rightarrow [d = h \cdot f(t,u); u = u + d; t = t + h; m = m + 1]$$

$$(y_{fin} = u) \wedge |y(t_{fin}) - y_{fin}| \leq \frac{p}{2q} \times \frac{\Delta}{n} \times \left(e^{q\Delta} - 1\right).$$

.

So, the control graph method and verification in KeYMaera with the provided contract are applicable here, and that graph is shown in Fig. 9.

This system can be proved with KeYmaera prover without user interaction after generating the HP code from the control-flow graph or by using any contract based prover after one will write a code of system for Euler method described in Fig. 8 and provide precondition, postcondition and invariant described in this section.

Nevertheless, for the more general problem of robots communicating, when each robot has this own function and needs to predict others robots moving, system specification will become not a simple task and verification of this definitely require automatic HP code generation by the model and developing of the system should be model-based. When robots start to use P2P communication, engineers should also use the Cyber part verification methods to prove the protocol, it is possible to use the methods described in the References section of this paper, and in future it is better to have all the methods integrated into the one tool.

9 Conclusion

In this paper firstly we addressed the problem of Cyber-Physical Systems (CPSs) modeling and validation. We state that Model-Driven Development (MDD) approach should be applied and there are some tools which support construction the CPSs with various level of abstractions without any code writing.

Next we discussed CPSs verification problem, and state that for a robust system design we need to use formal verification methods and tools. Because the CPSs need the Dynamic Logic to model the equations over the continuous time, it is hard to verify such systems with conventional verification tools designed for discrete-state systems. We state that right now there is a deficit of industrial-strong tools which provides verification of both the Cyber and the Physical parts of a CPSs.

Then we considered a sample CPS — a car braking example to show how to model and verify CPSs easier.

Finally we moved to the conception of cyber-physical BDI agents to show various problems with it. The Mars Rovers problem for robot-of-robot prediction can be numerically solved with using the Euler method, and this method helps to construct the considered verification strategy for this problem using our control-flow graph approach.

But for the problems that require BDI agents interoperations (for example, in a P2P manner) it is hard to describe and model the system behavior and provide the verification strategy. We think that a Model-Driven method to design and verify such types of systems should be developed.

The topic for the further research is to extend the control-flow graph approach to convert the existing engineering notations to an intermediate representation and then automatically create Hybrid Automata, proof obligation and verify them with the KeYmaera theorem prover or other suitable methods.

References

1. A. Platzer, "Differential dynamic logic for verifying parametric hybrid systems." in *TABLEAUX*, ser. LNCS, N. Olivetti, Ed., vol. 4548. Springer, 2007, pp. 216–232.
2. Platzer, André, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.
3. A. Platzer, "Logical Foundations of Cyber-Physical Systems," *Switzerland: Springer*, 2018.
4. T. A. Henzinger, "The theory of hybrid automata," in *Verification of Digital and Hybrid Systems*. Springer, 2000, pp. 265–292.
5. E. Hairer, C. Lubich, and M. Roche, *The numerical solution of differential-algebraic systems by Runge-Kutta methods*. Springer, 2006, vol. 1409.
6. W. Trench, *Elementary Differential Equations*. Thomson Learning, 2001. [Online]. Available: https://digitalcommons.trinity.edu/mono/8/
7. B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.
8. S. Staroletov, *Basics of Software Testing and Verification [in Russian]*. Lanbook, Saint Petersburg. ISBN 978-5-8114-3041-3, 2018.
9. L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.
10. D. Lozhkina and S. Staroletov, "An online tool for requirements engineering, modeling and verification of distributed software based on the MDD approach," *Some Journal*, 2008.
11. E. Kindler, "Safety and liveness properties: A survey," *Bulletin of the European Association for Theoretical Computer Science*, vol. 53, no. 268-272, p. 30, 1994.
12. G. A. Agha, "Actors: A model of concurrent computation in distributed systems." Massachusetts Inst Of Tech Cambridge Artificial Intelligence Lab, Tech. Rep., 1985.
13. V. E. Zyubin, "Hyper-automaton: A Model of Control Algorithms," in *IEEE International Siberian Conference on Control and Communications (SIBCON-2007). Proceedings.*, O. Stukach, Ed. Tomsk, Russia: IEEE, 2007, pp. 51–57. [Online]. Available: https://doi.org/10.1109/SIBCON.2007.371297
14. A. S. Rozov and V. E. Zyubin, "A hyperprocess-based approach in Arduino programming," *International Conference on Advanced Technology & Sciences (ICAT'15)*, 2015.
15. H. Moore, *MATLAB for Engineers*. Pearson, 2017.
16. P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.
17. J. Davis II, M. Goel, C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie *et al.*, "Overview of the Ptolemy project," ERL Technical Report UCB/ERL, Tech. Rep., 1999.
18. S. Staroletov, "Towards problems of cyber-physical systems verification while designing them with the model-driven approach," *16th International Scientific-Practical Conference of Students, Post-graduates and Young Scientists "Youth and Modern Information Technology" (YMIT)*, 2018.

19. A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.
20. G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.
21. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.
22. G. Frehse, "Phaver: algorithmic verification of hybrid systems past hytech," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 3, pp. 263–279, 2008.
23. L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, and T. Villa, "Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis," in *International Workshop on Reachability Problems*. Springer, 2012, pp. 79–91.
24. A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems (system description)," in *International Joint Conference on Automated Reasoning*. Springer, 2008, pp. 171–178.
25. T. Baar and S. M. Staroletov, "A control flow graph based approach to make the verification of cyber-physical systems using KeYmaera easier," *Modeling and Analysis of Information Systems*, vol. 25, no. 5, pp. 465–480, 2018.
26. F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-c: A software analysis perspective," *Formal Aspects of Computing*, vol. 27, no. 3, pp. 573–609, 2015.
27. C.-H. Cheng, T. Fristoe, and E. A. Lee, "Applied verification: The Ptolemy approach," *Some Journal*, 2008.
28. E. M. Clarke Jr, O. Grumberg, D. Kroening, and H. Veith, *Model checking*. Cyber-Physical Systems, 2018.
29. M. Takada, "Distributed systems: for fun and profit," 2013. [Online]. Available: http://book.mixu.net/distsys/
30. A. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms*, 2nd ed. Prentice-Hall, 2006.
31. M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd ed. John Willey & Sons, 2009.
32. R. F. R., J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*. MIT Press, 1995.
33. J. Ichikawa and M. Steup, "The analysis of knowledge." [Online]. Available: http://plato.stanford.edu/entries/plato-theaetetus/
34. C. Chappell, "Plato on knowledge in the theaetetus." [Online]. Available: http://plato.stanford.edu/entries/plato-theaetetus/
35. P. Dütting and A. Geiger, 2007. [Online]. Available: http://www.staff.science.uu.nl/˜leeuw112/msagi/mech_design.pdf
36. N. Shilov, N. Garanina, and E. Bodin, "Multiagent approach to a dijks-tra problem," in *CS&P'2010 Workshop on Concurrency, Specification and Programming*. Humboldt-Universitat zu Berlin, 2010.
37. S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
38. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, 2nd ed. Springer, 2012.