Time Specification, Modeling and Measurement in frame of Cyber-Physical System Applications Design

Miroslav Sveda
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
E-mail: sveda@ fit.vutbr.cz

Abstract—**This paper addresses the role, interpretation and the deployment of the notion "time" in distributed cyber-physical systems. Stemming from a brief state-of-the-art review, it discusses various possibilities how to specify, model and measure miscellaneous features of real time in the domain applications. The manuscript brings a simple, timing-oriented formal semantics of an example specification language and demonstrates the developed approach using case studies. The aim of the article is to select the fitting methods that enable to utilize the related specification and design approach for distributed cyber-physical systems applications.**

Keywords-cyber-physical system; time; temporal partial order; operational semantics; measurement.

I. INTRODUCTION

The integration of physical systems and processes with networked computing has led to the emergence of a new generation of engineered systems: Cyber-Physical Systems (CPS) [9]. Such systems use computations and communication deeply embedded in and interacting with physical processes to add new capabilities to physical systems. These cyber-physical systems range from small embedded applications, such as pace makers to large-scale huge systems, e.g. the international power-grid. Because computer-augmented devices are everywhere, they are a huge source of economic leverage. Embedded computers allow designers to add capabilities to physical systems that they could not feasibly add in any other way. By merging computing and communication with physical processes and mediating the way how to interact with the physical world [14], cyber-physical systems bring many benefits: they make systems safer and more efficient; they reduce the cost of building and operating these systems; and they allow individual machines to work together to form complex systems that provide new capabilities. By merging computing and communication with physical processes and mediating the way we interact with the physical world, cyber-physical systems bring many benefits: they make systems safer and more efficient, they reduce the cost of building and operating these systems, and they allow individual machines to work together to form complex systems that provide new capabilities.

This paper considers the orchestration of computing with physical processes. It argues that to realize its full potential, the core abstractions of computing need to be rethought to incorporate essential properties of the physical systems, most particularly the passage of time [13], [4].

The kernel of the paper consists of an explanation of the notion "time" in sections II and III, and of presenting the Asynchronous Specification Language (ASL) including its operational semantics for temporal partial order in sections IV and V. Next section discusses two case studies. The first case demonstrates using ASL for behavioral specification of lift cabin position measurement, and the second case introduces standards based time measurement settings with clock synchronization in a distributed system based on Internet.

II. TIME: STATE OF THE ART

Norbert Wiener in the Chapter 1, Newtonian and Bergsonian Time, of his book [22] distinguishes between reversible, Newtonian time of classical mechanics and irreversible time of cybernetics with definite past-future order fitting also such disciplines as meteorology, thermodynamics, statistical mechanics, and biology. Physicists perfect this notion into thermodynamic, psychological, and cosmological time arrows that point in the same direction [5].

Various disciplines utilize several notions of time. For instance, Christie and Halpern in [27] and [28] discuss the concept of time for history and anthropology by the following way: "Nowadays, historical descriptions are usually concerned with unique, sequential events within a linear time frame linked to some combination of preceding political, religious, economic, and social events. These events, being unique, are therefore not readily predictable. By contrast, socio-cultural anthropological accounts tend to focus on, as the center of concern and the ultimate reality, happenings in cyclical temporal dimension-events, which are recurrent and can be foreseen within the context of the system. Just a cyclical time is a dominant factor in anthropology--specifically bounded linear time segments, within a given spatial dimension, are the key orienting factors in a history curriculum." Apparently, they address an ordering of time domain that fits either history or anthropology.

Contemporary Cybernetics and Computer Science deal -- in frame of their branches such as Artificial Intelligence, Systems Theory, or Software Engineering -- with various concepts and refinements of directed time. Current research in this domain focuses on warped time in context of multi-time partial differential equations, widely-separated time scales or multi-time solitons [29], and on two-dimensional systems theory in frame of geometric approach [30].

While the concept of global time dominates in current specification techniques, the concept of local time is also investigated. Several suggestions appeared over the past few years to extend the expressive power of real-time temporal or modal logics to handle local-timing constraints. For example, Wang, Mok, and Emerson [31] devise the Asynchronous Propositional Temporal Logic for specification and verification of distributed hard real-time systems; actually, the language deals explicitly with multiple local clocks. Of course, there are some theoretical limitations of timing in such languages: Alur and Henzinger [32] prove that even two independent time functions, which can represent the reading of two different clocks, lead to undecidability of a pertinent real time temporal logic provided that no restrictions on the locality of time references within atomic formulae are defined. Also, because of close relationship between modal or temporal logics and various types of transition systems, the same can be claimed as for fitting timed versions of the latter group of models. Besides, real-time process algebras were adapted to deal with other than totally ordered time domains. Jeffrey [33], Murphy [34], and Baeten and Bergstra [35] designed various calculi capable to describe some relativistic phenomena. While Jeffrey follows observing paradigm to deal with light-cones-based space-time observations, Murphy applies testing to obtain both timing and causality as relations on space-time. Baeten and Bergstra [36] extend their Real Time ACP to take space coordinate into account and so develop a relativistic calculus. Complementary, in [35], they limit the use of space to a finite set of locations and combine a state operator and classical, i.e. non-relativistic, real space process algebra.

Probably the closest conception to the time model presented in this paper is supported by Janssen, Poel, Xu, and Zwiers [37] with the shared variable language possessing real-time constructs for temporal order and layering principle of casual order. Dealing explicitly with local time assessment, they introduce a clock assumption that governs local clocks and enables to model not only perfect isochrony but also weaker types of synchronization in frame of temporal partial order. Dated synchronous languages, such as ESTEREL and LUSTRE, also pretend to express real-time constraints without making reference to a global physical notion of time: they use the so called multiform time that can refer to different resources. Nevertheless in the completely synchronous world, the notion of physical metric time is replaced by simple order and simultaneity so that the local clocking mechanisms are in fact derived from some common global ticks. The same model of multiform time is utilized for Synchronized Elementary Systems where, moreover, such notions as inertia (the minimal number of occurrences of some time reference event before a pre-condition holds) and latency (the maximal number of that before the effective occurrence of an event "ready to occur") can be expressed [38].

Basic meanings of the term "time" can be introduced in the following complementary couples: physical/logical, absolute/relative, global/local. To be more precise, we consider an event domain, E, and a time domain, T, such that instead of viewing the precedence relation "to causally affect" on events we use members of a time domain to mark the members of the event domain to introduce a temporal order [12]. Especially, the physical time means that passing of time is the primary cause for anything to happen; actually, it denotes counting cycles of a physical, strictly periodic process [1]. The logical time means that time passes only because something happens -- it respects order of events only [8]. The absolute time means that a reference is established in relation to a unique event for a given system; evidently, it relates to some origin of date/time, see e.g. [10]. The relative time means that a reference is established in relation to an arbitrary selected event in the given system; clearly, it relates to time intervals. The global time means that the time is considered to be valid for the whole (distributed) system while the local time means that the time is valid for a part of the (distributed) system.

Various disciplines utilize several notions of time. To be more complete, we should specify time models by more attributes than precedence relation. Models of time can be classed, see e.g. [8], according to individuals (points, intervals), order (partial order, branching towards future, linear), boundedness (unbounded, beginning, ending), local structure (discrete, dense, continuous) and global structure (connectedness, homogeneity). Whereas synchronous models of computation regard all concurrent activities happen in a lock-step, asynchronous models are not restricted in this sense. They can be treated as interleaving models of computation, which sequentialize simultaneous actions non-deterministically, or as true concurrency models of computation, which impose only a partial ordering between actions. By the way, various models were designed aiming to describe also some relativistic phenomena, see e.g. [3].

III. SPACE AND TIME DOMAINS

Real-time systems are not only event driven but, moreover, they must respect the timing of their environmental processes. Usually, specification tools consider unique global timing for distributed real-time systems. Focusing on formal specifications in that field, the paper pleads for another clocking conception: to time by multiple local clocks. Local time, originally introduced to Computer Science by Lamport [11], is considered to be unique only for a predefined part of a distributed system. For the current paper, it means the time of a local physical clock based on some periodic physical oscillation whose frequency suits the measurement of the duration of local process actions.

This section selects and narrows some ideas from [8], and [24] focusing on local time. The treatment of event-time relationships resembles to the approach presented in [15]; however, time domain is shifted from total order to partial order in this case. Hence, we consider an event domain, E, and a time domain, T, such that instead of viewing the precedence relation "to causally affect" on events we use members of a time domain to mark the members of the event domain to introduce a temporal order.

For each of the domains E and T there are two possibilities how to choose domain elements: points and intervals. To preserve simplicity, we select points for both domains. Consequently, events can be interpreted as changes of system states and members of time domain as time instants. In this case, timing of events is mapping $E \rightarrow T$. Actions with non-zero duration can be described by their starting and ending points that require individual timings. Point structures of domains E and T simplify introduction of partial order in general. Local time concept requires employing a partial order consistent with locality either of events or of timing. There are at least two natural possibilities how to introduce a timed partial order on events: (a) to define locality as an equivalence relation on events Loc = (E, ~) and then, for each class of that equivalence to specify a separate linear time, i.e. to use multiple time lines (see similar conception in [2]); or (b) to connect locality explicitly with temporal partial order $(Loc \times T, \sim)$, see e.g. [23]. From the application viewpoint, both possibilities correspond to the same relation: for case (a), temporal partial order is induced on T by manifold mapping; for case (b), partially ordered time generates a decomposition of event domain into classes so that events in each class are mutually comparable by linear temporal order. Nevertheless, we prefer the case (a).

Koymans [8] distinguishes three local temporal structures: discrete, dense, and continuous. The same can be applied to space or even space-time coordinates, see e.g. [3]. We prefer scalable discrete time structure and fix discrete finite space structure in form of finite set of locations.

An implicit time domain of a system process respects internal events (changes in the state) of that process. An explicit time domain, on the other hand, consists of events that are not produced in the process, but which bear an observable temporal relation of the local process [6]. Both types of timing can be considered as either internal to the local process or external to the remote processes (e.g. environmental processes). Evidently, implicit timing suffices only for a synchronous system timed by a common global clock or for a system driven by only one sequential process while real-time (asynchronous) distributed systems require explicit time domains. In accordance with Holt [7], a model of real-time systems is natural if its internal time corresponds well with the external, physical time of the environment. However, different timing mechanisms rule various parallel environmental processes. In addition, distributed applications consider a distinctive, locally measured time for each node. A useful time model, therefore, must conform with external events as well as to internal timing, and it should provide unambiguous semantics for a specification and implementation of real-time distributed systems.

IV. OPERATIONAL SEMANTICS FOR TEMPORAL PARTIAL ORDER

The particular behavior of a non-Zeno, discrete real-time system can be described by an infinite sequence of pairs of state s_i and corresponding time t_i [2]:

$$P: \quad (s_0, t_0) \rightarrow (s_1, t_1) \rightarrow (s_2, t_2) \rightarrow \ldots$$

Different models of time interpret the time component, t, of the system behavior, P = (s, t), in different ways. While interval models of time associate each state with its duration over time, clock models stamp observations of the node state with time instants. To characterize asynchronous systems, whose node state changes can be arbitrarily close in time, analog-clock models record the exact time of every state. By contrast, digital-clock models measure the time of a state only with finite precision, approximating a dense time domain by a sequence of discrete values — the time between successive states may remain the same or may increase by an arbitrary amount. For a distributed system, its state space can be decomposed into the state spaces of its nodes 1, 2, ..., n:

$$\begin{array}{l}
{}^1P: \ ({}^1s_0, {}^1t_0) \rightarrow ({}^1s_1, {}^1t_1) \rightarrow ({}^1s_2, {}^1t_2) \rightarrow \ldots \\
{}^2P: \ ({}^2s_0, {}^2t_0) \rightarrow ({}^2s_1, {}^2t_1) \rightarrow ({}^2s_2, {}^2t_2) \rightarrow \ldots \\
\qquad\qquad \ldots \\
{}^jP: \ ({}^js_0, {}^jt_0) \rightarrow ({}^js_1, {}^jt_1) \rightarrow ({}^js_2, {}^jt_2) \rightarrow \ldots \\
\qquad\qquad \ldots \\
{}^nP: \ ({}^ns_0, {}^nt_0) \rightarrow ({}^ns_1, {}^nt_1) \rightarrow ({}^ns_2, {}^nt_2) \rightarrow \ldots
\end{array}$$

In this case, additional attributes of time clarify the nature of the time component, ${}^jt$, of the node j's behavior, ${}^jP = ({}^js, {}^jt)$: real-time distributed architecture enriches models of time by considering the number of time lines. A single time line suffices for global clocks while multiple time lines support independent local clocks. Accordingly, the values ${}^jt_i$ and ${}^kt_i$ are either i-th readings of global time, t, in nodes j and k, or i-th readings of local times ${}^jt$ and ${}^kt$ in nodes j and k. To respect the implementation viewpoint, distributed applications consider for each node a distinctive local time, i.e. the time of a local physical clock that suits to measuring duration of local process actions.

Local time represents a concept of physical timing; still, its semantics can be derived from logical time and a physical generator of periodic events. In his pioneer work [11], Lamport defines logical time in a distributed system as a partial ordering of events in the system. Similarly for the purpose of this paper, time ordering of events in a system S is specified by a minimal partial-order relation $\rightarrow$ ("precedes") on events, which satisfies the following four conditions:

- if A and B are events in the same process and A is executed before B, then $A \rightarrow B$ (the term "process" means sequential ordering of internal events);
- if A is the transmission of information by one process and B is the receipt of that information in S, then $A \rightarrow B$ (communication proceeds in non-zero time);
- if $A \rightarrow B$ and $B \rightarrow C$ in S, then $A \rightarrow C$ (transitivity); and
- for any event A of S, $A \rightarrow A$ does not hold ($A \nrightarrow A$) (antireflexivity).

The totalization of the reflexive closure of the relation $\rightarrow$ is the relation $\Rightarrow$ ("precedes or is concurrent"):

- if $A \nrightarrow B$ in S, then $B \Rightarrow A$.

An eventcount, E, is an object that counts the number of events of a specific type that have occurred during the execution of the system. Each such event occurrence invokes the operation ADVANCE(E): E := E + 1. The external operation AWAIT (E, σ) suspends the calling process until the value of the eventcount, E, is at least σ — the call AWAIT (E, σ) usually resets the current value of E, so the relative timing is possible. The external and internal operations interplay in the following way:

- if WE is the execution of the AWAIT operation in the form AWAIT (E, σ), then there are at least σ members of the set $\{AE \mid AE$ is the execution of ADVANCE (E) and $AE \Rightarrow WE\}$.

An eventcount can monitor the external events of a class that represents local physical timing of a distinctive part of the system environment. Also, periodic events implemented by an internal timer/counter circuit can advance an eventcount that tracks local-time clock events. So, the local-time model relates internal and accessible external clocking while internal local times in different nodes of a distributed system flow independently, without regular synchronization.

V. ASYNCHRONOUS SPECIFICATION LANGUAGE

The designed process-oriented procedural specification language includes primitives related to synchronization, timing, and communication. Hence, the specification of a system logical structure employs sequential processes, communicating asynchronously by message passing in distributed surroundings. A process represents the sequence of statements executed at a node of a distributed system. True concurrency with maximal parallelism is supposed: each process drives its own node; if a process is suspended, its node remains idle. Moreover, timing mechanisms of the processes are expressed with the help of local timers, using properly chosen time scales.

From the syntax viewpoint, the language for process specifications can be surveyed as an extended Pascal. The most important added primitives relate to process specification, timing, communication, and control structure.

```
process name (is: list_of_s_inputs; os: list_of_s_outputs;
      ic: list_of_m_inputs; oc: list_of_m_outputs):        ...
      ...   endprocess;
wait (_, timeout); wait (event, _);
      wait (event, timeout, test);
send (message, destination);
loop ... [... when <cond> action ... exit]* ... endloop;
```

Each of asynchronous processes can be equipped by its individually timed local clock, can receive messages through input buffer, and can send messages to other, directly or indirectly addressable processes. Process header contains in parentheses lists labelled by *is*, *os*, *ic*, and *oc* that act as the interface with the process' environment. The language distinguishes between signal inputs or outputs, which denote un-buffered events carrying either value or signalling their occurrence, and message inputs or outputs as typed asynchronous channels between couples of processes. Those signals and messages declare the inter-process synchronization and communication, which operations are driven by the statements *wait (event, _)*, *wait (event, timeout, test)*, and *send (message, destination)*.

The primitive *wait (_, timeout)* suspends a process for the interval defined by the value *timeout*. Operational semantics can be obtained through the eventcount abstraction introduced above. In this case, an event is every tick of the local clock, so the related operation is AWAIT (local_ticks, timeout_value). For the primitive *wait (event, _)*, which suspends a process until specified event (external signal or message) appears, the model operation is AWAIT (event_type, 1). Semantics of the combined statement *wait (event, timeout, test)* requires two eventcounts: the first anticipates the specified event and the second, with a lower priority, monitors the local clock. The reason of process activation can be checked through the value of the logical variable *test*: when the value is true, the *event* occurred within the interval *timeout*.

The primitive *send (message, destination)* implements asynchronous communication with non-blocking semantics. To respect different local clocks, the information transfer is controlled by a special clocking that is common for the source and the destination; however, the nodes communicate asynchronously by message passing through an input buffer at the destination. The input of a message induces the event for the related operation AWAIT (message, 1). If any synchronization is required, it must be described explicitly using confirmation and the *wait* statements.

The control structure primitives *loop ... endloop* delimit an indefinite cycle, which is exited upon a true result of testing the condition following the primitive *when*. Consequently, the statements, which occur between the primitives *action* and *exit* and which follow the *endloop* primitive, are executed. This combined statement enables to extend the language with additional control structures by simple macro-like text replacements.

```
if <cond> then <s1> else <s2> fi;
~loop when <cond> action <s1> exit <s2> when true exit
endloop;
timeloop (timeinterval) ... endloop;
~loop ... wait (_, interval) endloop;
```

Actually, the control structure *timeloop (timeinterval) ... endloop* specifies an isochronous loop, which is periodically initiated whenever the *timeinterval* expires and which can be exited like the indefinite cycle. The operation AWAIT (local_ticks, timeinterval_value) defines the exact operational semantics of timing these initiations.

VI. CASE STUDIES

A. *Lift cabin position measurement for drive control*

An incremental measurement device for position evaluation of a lift cabin in a lift shaft facilitates the demonstration of the local time specification and design. Incremental measurement means the recognition and counting of rectangular impulses that are generated by an electromagnetic or photoelectric sensor/impulse generator, which is fixed on the bottom of the lift cabin and which passes equidistant position marks along the shaft. The device is dedicated to support both (a) cabin position controls by communicating the actual position value to a drive controller and (b) lift maintenance by displaying the actual position. For the sake of safety requirements, the device has to be equipped by fault detection mechanisms.

The device communicates with its environment through two classes of input/output variables. The first class describes an interface with the impulse generator and drive controller as inputs. The timing for this class is driven by maximal cabin speed and by precision requirements for position measurements. The second class contains two output variables: (1) communication output to a drive controller, which is timed according to the control requirements, and (2) display with position indication as a human interface, which must respect human-physiology timing constants. For the demonstration, we restrict our attention to the display output only. So, the first input variable, I, contains the values 0 or 1 that are altered with frequency equivalent to the cabin speed. The second input variable, D, contains the values "up," "down," or "idle." The auxiliary input variable, level, serves the initial position synchronization. The output variable, P, indicates the absolute position of the cabin in the shaft.

The first case study consists in the logical structure description of the two-level structure, where higher level behaves as an event-driven component and lower level behaves as time-evolving interconnected component. The behaviour of the higher level component can be described by the following state sequence:

initialization → position_indication → fault_indication

The behaviour of the lower level can be described by three communicating, individually timed automata. The first automaton models the impulse detector, timed by its local clock that defines a sampling interval. This interval must conform not only to the maximal speed and the distance of position marks but also to a pattern of samples for impulse recognition, depending on the electro-magnetic interference characteristics of the environment. Let the fitting pattern, skipping possible transient fault states, is represented by '1100'. Then adequate behaviour for impulse recognition can be described by the following sampling sequence with regular periodic timing:

$$q_1 \overset{\text{inp=1}}{\rightarrow} q_2 \overset{\text{inp=0}}{\rightarrow} \ldots\ q_2 \overset{\text{inp=1}}{\rightarrow} q_3 \overset{\text{inp=1}}{\rightarrow} \ldots\ q_3 \overset{\text{inp=0}}{\rightarrow} q_4 \overset{\text{inp=1}}{\rightarrow} q_4 \overset{\text{inp=0/IMP}}{\rightarrow} q_1$$

The information about detected impulse is sent to the counting automaton, which can also access the indication of the cabin movement direction through the variable D. The counting automaton communicates the position value to the display automaton. The display refreshment subsides to a timing mechanism dependent on the physiologic constants of human sight. For the sake of fault-detection requirements, the impulse generator and transfer path are doubled. Consequently, a second, identical impulse detector automaton appears necessary. The subsequent automaton is the reversible counter, which starts with the value (h+l)/2 and increments or decrements the value according to the "impulse detected" outputs from the first or second recognition automaton. Overflow or underflow of the preset values of h or l indicates an error.

The description of the system's logical structure in Asynchronous Specification Language employs concurrently running parallel processes with true concurrency, communicating asynchronously by message passing. From the above mentioned behavioral specifications by communicating automata, the logical structure design proceeds directly by describing automata transitions by means of processes described by the above introduced specification language. In this step, the timing mechanisms are expressed with the help of local timers with properly chosen time scales.

```
process detection (is: I0, I1, D; os: error; oc: counter):
type counter = process;
type message = record null end;
type direction = (up, down, idle);
var D: direction;
var impulse: message;
var error: boolean;
var q0, q1, count, l, h, sample_interval: integer;
var in0, in1, I0, I1: binary;
loop  q0:= 1; q1:= 1; count:= (h+l)/2; wait (D=idle, _);
write(false, error);
      wait (D<>idle, _);
      timeloop (sample_interval)
       read(in0, I0); read(in1, I1);
       if q0 <= 2 then if in0 = 1 then q0 := q0 + 1 fi
           else if in0 = 0 then q0 := q0 + 1 fi fi;
       if q1 <= 2 then if in1 = 1 then q1 := q1 + 1 fi
           else if in1 = 0 then q1 := q1 + 1 fi fi;
       if q0 >= 4 then q0 := 1; count := count - 1;
      send (impulse, counter) fi;
       if q1 >= 4 then q1 := 1; count := count + 1 fi;
       when l > count or count > h action write(true,error) exit;
      endloop;
endloop
endprocess;

process counter (is: D, level; os: error; ic: impulse; oc:display):
type display = process;
type message = record null end;
type direction = (up, down, idle);
var D: direction;
var impulse: message;
var level, lvl, maxlvl, minlvl: integer;
var error: boolean;
loop  wait (D=idle,_); read(lvl,level); write(false,error);
      wait (D<>idle);
      loop wait (impulse,_); if D = up then lvl := lvl + 1
                                 else if D = down then lvl := lvl - 1;
       when (lvl > maxlvl) or (lvl < minlvl)
                                 action write (true, error) exit;
       send (lvl, display);
       when D = idle action exit;
      endloop;
endloop;
endprocess;
```

The reviewed design example complies with such decisions as incremental measuring, periodic sampling of impulses, and comparing the outputs of doubled incremental detector and transfer path. To implement the overall fault detection capability, it is necessary also to check the counter and the display. This task can be realized e.g. by a watchdog process that can relate input impulses from the sensor/impulse generator to the changes of displayed value; the appropriate automaton performs again reversible counting.

B. Communication time measurement compatible with IEEE 1588

Cyber-physical systems can be highly connected and integrated in multiple ways, even across business operations and domain boundaries. Achieving effectively networked, cooperating, and human-interactive systems will be an integral factor in the adoption of such systems in the future. Some of the key questions to be considered include what is needed to enable streamlined and predictable development, deployment, and evolution of networked and integrated cyber-physical systems, particularly as systems become interconnected with legacy systems and across industry boundaries.

This case study addresses important topics associated with the measurement of data communication delays in computer networks. The entire application consists in developing one-way delay measurement method and related support for Internet environment including monitoring and comparison of computer clocks, pulse-per-second signal processing, time server with guaranteed accuracy concepts and high accuracy time-stamping implementation. For the demonstration of timing issues, only a small subset of the complete task is presented. Because measurement of one-way communication delay is a two-point measurement task, it requires either synchronized clocks or clocks with known time offset in both points of measurement.

Any distributed cyber-physical system requires maintaining time synchronization in quality depending on a particular application domain. Many current distributed systems employ Ethernet or even full TCP/IP protocol stacks for communication. In such setting, the following two protocols are usually used for clock synchronization: Network Time Protocol (NTP) and Precise Time Protocol (PTP) [25]. This section presents an approach to experimental performance evaluation of clock synchronization within a distributed environment consisting of nodes running a commodity real-time operating system. The results of measurement can provide a hint on the degree of synchronization that can be practically achieved among the set of distributed devices in the considered configuration.

Important characteristics of communication networks for respecting real-time requirements deal namely with Transit Delay and Delay Variation with respect to actual Network Load. The setting considers packet switching networks, which in contrast to TDMA or similar techniques, cannot guarantee constrained transfer time. In fact, actual Transit Delay and Delay Variation depend on actual Network Load:

- Packet Transit Delay (PDT) - means packet delay due to data signal holdup, due to data processing by input or output in active devices, and during the period when packet is stored in buffers.
- Packet Delay Variation (PDV) - means the difference of individual packet’s PDTs.
- Network Load (NL) – is directly influencing both PDT a PDV. Measuring NL and related values of PDT and

PDV enables us to define functional dependencies for a particular network.

Evidently, this measurement considers relative physical timing that facilitates clock synchronization using NTP and PTP protocols with precision better then 1 ms without any additional hardware support.

VII. CONCLUSIONS

This paper stems from the author's research projects with partial results published in [16] ... [21]. The current paper addresses the role, interpretation and the deployment of the notion "time" in distributed cyber-physical systems. It discusses various possibilities how to approach such modeling and selects the fitting one, which enables to utilize the related specification language ASL in the domain applications specifications, modeling and measurements. The preliminary version of the paper was presented at the EUROPMENT SCSI 2013 Conference [26].

ACKNOWLEDGMENT

This project has been carried out with a financial support from the Czech Republic state budget by the IT4Innovations Centre of Excellence, EU, CZ 1.05/1.1.00/02.0070CEZ and by the MMT project no. MSM0021630528: Security-Oriented Research in Information Technology, by the Technological Agency of the Czech Republic through the grant no. TA01010632: SCADA system for control and monitoring RT processes, by the Ministry of Industry and Trade of the Czech Republic through the grant no. FR-TI1/037: Automatic Attack Processing and by the Brno University of Technology, Faculty of Information Technology through the specific research grant no. FIT-S-11-1: Advanced Secured, Reliable and Adaptive Information Technologies.

The author acknowledges contributions to the presented work by his colleagues Ondrej Rysavy, Petr Matousek, Jaroslav Rab, Vladimir Vesely, Matej Gregr, Libor Polcak and Patrik Halfar from the Faculty of Information Technology, and Radimir Vrba from the Faculty of Electrical Engineering and Communication.

REFERENCES

[1] D.W. Allan and N. Ashby and C. Hodge. Time in the Space Age. IEEE Spectrum vol.35, no.3, 2012, pp.42-51.

[2] R. Alur and T.A. Henzinger: Logics and Models of Real Time: A Survey. In: J.W. de Bakker et al. (eds.), Real-Time: Theory in Practice. Springer-Verlag, LNCS 600, 1992, pp.74-106.

[3] J.C.M. Baeten and J.A. Bergstra: Real space process algebra. In J.C.M. Baeten and J.F. Groote (eds), Concur 91, Springer-Verlag, LNCS 527, 1991, pp.96-110.

[4] J.C. Eidson, E.A. Lee, S.Matic, S.A. Seshia, J.Zou: A Time-Centric Model for Cyber-Physical Applications, MoDELS 2010, ACES-MB Workshop Proceedings, Oslo, Norway, October 4, 2006, pp.21-35.

[5] S.W. Hawking: A Brief History of Time--From Big Bang to Black Holes. Bantam Books, New York, 1988.

[6] A. Goswami and M. Joseph: Defining Time Domains for Computation. In: H.S.M. Zedan (ed.), Real-Time Systems, Theory and Applications. North-Holland, Amsterdam, 1990, pp.49-61.

[7] C.M. Holt: Intervals as Time Lattices. In: H. S. M. Zedan (ed.), Real-Time Systems, Theory and Applications. North-Holland, Amsterdam, 1990, pp.63-79.

[8] R. Koymans. (Real) time: a philosophical perspective. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg (eds), Real-Time: Theory in Practice, pp.353-370. Springer-Verlag, 1992. LNCS 600.

[9] B.H. Krogh, E. Lee, I. Lee, A. Mok, R. Rajkumar, L.R. Sha, A.S. Vincentelli, K. Shin, J. Stankovic, J. Sztipanovits, W. Wolf, W. Zhao: Cyber-Physical Systems, Executive Summary, CPS Steering Group, Washington D.C., March 6, 2008. [http://www.nsf.gov/pubs/2008/nsf08611/nsf08611.htm]

[10] M. Kudlek: Calendars and Chronologies. In C. Freksa, M. Jantzen, and R. Valk (eds), Foundations of Computer Science. Springer-Verlag, 1997. LNCS 1337.

[11] L. Lamport: Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM, vol.21, July 1978, pp.558-565.

[12] L. Lamport: Temporal Logic of Actions. ACM Transactions on Programming Languages and Systems, vol.16, no.3, 1994, pp.872-923.

[13] E.A. Lee: Computing Needs Time, Communications of the ACM, vol.52, no.5, May 2009, pp.70-79.

[14] E.A. Lee: CPS Foundations, DAC'10, Anaheim, California, USA, ACM, June 2010, pp.737-742.

[15] S.E. Pomares Hernandez, J.R. Perez Cruz, M. Raynal, From the Happened-Before Relation to the Causal Ordered Set Abstraction, J. Parallel Distributed Computing, vol. 72, Feb. 2012, pp.791–795, doi:10.1016/j.jpdc.2012.02.015.

[16] O. Rysavy, M. Sveda, R. Vrba: A Framework for Cyber-Physical Systems Design - A Concept Study, Proceedings the Sixth International Conference on Systems ICONS 2012, Saint Gilles, Reunion Island, FR, IARIA, 2012, pp.79-82.

[17] M. Sveda: Local Time for Formal Specification of Networked Embedded System, In: WSEAS Transactions on Computers, Vol. 2, No. 1, 2003, Athens, GR, pp.4-9.

[18] M. Sveda: Rapid Prototyping of Networked Embedded Systems, In: Proceedings of the IEEE International Conference and Workshop on the Engineering of Computer-Based Systems 2003, Huntsville, AL, US, IEEE CS, 2003, pp.125-132,

[19] M. Sveda: A Design Framework for Internet-Based Embedded Distributed Systems, In: Proceedings of the International IEEE Conference and Workshop ECBS'2004, Los Alamitos, California, US, IEEE CS, 2004, pp.113-120

[20] M. Sveda, R. Vrba: An Embedded Application Regarded as Cyber-Physical System, Proceedings of the Fifth International Conference on Systems ICONS 2010, Les Menuires, FR, IEEE CS, 2010, pp.170-174.

[21] M. Sveda, R. Vrba: A Cyber-Physical System Design Approach, Proceedings of the Sixth International Conference on Systems - ICONS 2011, St. Maarten, AN, IARIA, 2011, pp.12-18.

[22] N. Wiener. Cybernetics or Control and Communication in the Animal and the Machine. Wiley, New York, 1948.

[23] S. Ying et al.: Foundations for Innovation in Cyber-Physical Systems, Workshop Report, Energetics Incorporated, Columbia, Maryland, US, January 2013.

[24] J. Zwiers: Layering and Action Refinement for Timed systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg (eds.), Real-Time: Theory in Practice Springer-Verlag, LNCS 600, 1992, pp.687-723.

[25] IEEE 1588-2002: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, IEEE, 2002.

[26] M. Sveda: Time in Cyber-Physical Systems - Specifications, Modeling and Measurements, Proceedings of the 2013 International

Conference on Systems, Control, Signal Processing and Informatics SCSI 2013, pp.413-419.

[27] J.M. Halpern: Interpreting the Past--Time perspectives and Social History. Studia Ethnologica (Zagreb), Vol.3, 1991, pp.85-99.

[28] L. Christie and J.M. Halpern: Temporal Constructs and Inuit Mental Health. Soc. Sci. Med. Vol.30, 1990, pp.739-749.

[29] C. Udriste: Multitime Solitons and Their Maple Animation, Proceedings of the 2013 International Conference on Systems, Control, Signal Processing and Informatics SCSI 2013, pp.23-24.

[30] V. Prepelita: Invariant Subspaces and Structural Properties of 2-D Control Szstems, Proceedings of the 2013 International Conference on Systems, Control, Signal Processing and Informatics SCSI 2013, pp.25-26.

[31] F. Wang and A.K. Mok and E.A. Emerson: Distributed Real-Time System Specification and Verification in APTL. ACM Trans. SEM, Vol.2, 1993, pp.346-378.

[32] R. Alur and T.A. Henzinger: Logics and Models of Real Time: A Survey. In: J.W. de Bakker et al. (eds.), Real-Time: Theory in Practice. Springer-Verlag, LNCS 600, 1992, pp.74-106.

[33] A. Jeffrey: Abstract timed observation and process algebra. In J.C.M. Baeten and J.F. Groote, editors, Concur 91, LNCS 527, Springer-Verlag, 1991, pp.332-345.

[34] D. Murphy: Testing, betting and timed true concurrency. In J.C.M. Baeten and J.F. Groote, editors, Concur 91, Springer-Verlag, 1991, pp.439-454..

[35] J.C.M. Baeten and J.A. Bergstra: Real space process algebra. In J.C.M. Baeten and J.F. Groote, editors, Concur 91, LNCS 527, Springer-Verlag, 1991, pp.96-110.

[36] J.C.M. Baeten and J.A. Bergstra: The state operator in real time process algebra. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, Real-Time: Theory in Practice, Springer-Verlag, LNCS 600, 1992, pp.107-123.

[37] W. Janssen and M. Poel and Q. Xu and J. Zwiers: Layering of real-time distributed processes. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, Formal Techniques in Real-Time and Fault-Tolerant Systems Springer-Verlag, LNCS 863, 1994, pp.393 417.

[38] C. Andre. Delays in synchronized elementary net systems. In G. Rozenberg, editor, Advances in Petri Nets, Springer-Verlag, LNCS 524, 1991, pp.1-28.