A Real-Time Semantics for the IEC 61499 standard

Per Lindgren*, Marcus Lindner*, Andreas Lindner* and Valeriy Vyatkin*[†] David Pereira[‡] and Luís Miguel Pinho[‡]

*Luleå University of Technology

[‡]CISTER / INESC TEC, ISEP

Email: {per.lindgren, marcus.lindner, andreas.lindner, valeriy.vyarkin}@ltu.se

Email: {dmrpe, lmp}@isep.ipp.pt

[†]Aalto University, Helsinki, Finland

Abstract—The IEC 61499 standard provides an executable model for distributed control systems in terms of interacting function blocks. However, the current IEC 61499 standard lacks appropriate timing semantics for the specification of timing requirements, reasoning on timing properties at the model level, and for the timing verification of a specific deployment.

In this paper we address this fundamental shortcoming by proposing Real-Time-4-FUN, a real-time semantics for IEC 61499. The key property is the preservation of non-determinism, allowing us to reason on (and verify) timing properties at the model level without assuming any specific scheduling policy or stipulating specific order of execution for the deployment. This provides for a clear separation of concerns, where the designer can focus on properties of the application prior to, and separately from, deployment verification. The proposed timing semantics is backwards compatible to the current standard, thus allow for re-use of existing designs. The transitional property allows timing requirements to propagate to downstream sub-systems, and can be utilized for scheduling both at device and network level.

Based on a translation to RTFM-tasks and resources, IEC 61499 the models can be analyzed, compiled and executed. As a proof of concept the timing semantics has been experimentally implemented in the RTFM-core language and the accompanying (thread based) RTFM-RT run-time system.

I. INTRODUCTION

The IEC 61499 standard provides an executable model for distributed control systems in terms of interacting function blocks (FBs). The execution model is event-driven (asynchronous), where triggering events may be associated with data. This allows for non-deterministic execution (which is plausible as actual scheduling can be tailored to best fit the underlying platform(s), network media etc., while correctness (as defined from the underlying semantics) is a model property.

Non-determinism, by itself, allows for many *correct* traces of a given input (and state), and can be observed (and reasoned on) using methods such as model-checking [1], [2]. This allows separation of concerns, where validating correctness of the model per-se (reasoning on the properties of the specification) can be done *prior to*, and *separately from*, validating that a specific deployment (implementation) meets the specification. In this way, the concern of the application designer is the specification and composition of models, where as s/he can rely on automatic methods (as a concern of the tool vendor) to validate that the deployment, (i.e., underlying platforms, run-time systems, networks, etc.), collectively meet the specification. Separation of concerns is a key property to dealing with systems of complexity, such as those emerging in industrial automation (where we actually face systems-of-systems).

However, the current IEC 61499 standard lacks appropriate semantics to natively specify intended timing behavior requirements. Instead, timing properties *emerge* from the implementation.

Related to real-time execution IEC 61499 model we find the work on TORERO [3]. Computations (actions) are implemented following the Java Real-Time Specification, while per-event timing specification is given using the IAONA Real-Time Classes. In the work of Zoitl [4], timing specification is expressed in terms of traditional notions of deadlines, inter-arrival and execution times. In comparison to [3], timing specifications in [4] are limited to *source* events. Another recent approach to real-time execution of IEC 61499 takes the outset from a Time-Complemented Event-Driven (TCED) control model [5]. Pang et. al. demonstrates that a TCED systems can be implemented following IEC 61499.

The related industrial standards IEC-61131/60848 and their underlying Sequential Function Charts (SFCs) and Grafccets have been extensively studied. In [6] an un-timed formalization of Grafccets in terms of Stable Location Automaton is proposed. The text nicely lays out the syntactic and semantic differences between the event driven Grafccet and the scan based (PLC) SFC execution model. A unified timed semantics is presented in [7]. Timing properties are expressed using 5 primitives (L, D, SL, SD, DS), each one with a unique semantics. It is found in [7] that the SFC semantics is ambiguous in the standard, and that a clear definition of well-formedness is lacking and open for interpretation.

In this paper we address this fundamental shortcoming by proposing Real-Time-4-FUN, a real-time semantics for the IEC 61499. We associate each event with a *Permissible Execution Window* (PEW), defined by its *baseline* (indicating the absolute arrival time), and a relative *deadline* (defining the timing requirement). Timing information is transitional (inherited), allowing a chain of events to operate under a common PEW (i.e., end-to-end timing requirement). Moreover, an emitted event can be associated a relative baseline offset, postponing its (downstream) arrival. This allows for expressing complex, yet statically analyzable, timing patterns free of accumulated drift (allowing e.g., delays and periodic behavior as trivial cases). Similarly to [3] we allow for timing specification of arbitrary events. While our timing semantics share similarities to [4] we propose a transitional and compositional semantics by means of baseline inheritance and offsets.

The timing semantics are influenced by previous work on the Timber language [8] and the lightweight TinyTimber [9] C-code programming model. The experimental implementation is based on the original work on the RTFM-core language [10].

II. BACKGROUND

A. IEC 61499

The IEC 61499 standard [11] provides a non-deterministic executable model for distributed control systems in terms of interacting function blocks. The executions semantics is informally defined, and thus subject to interpretation (as no official reference implementation is present). For the purpose of the presented work, we briefly summarize key features of the standard. For a comprehensive overview see [12].

1) Design Elements: Function Block (FB) types:

- Basic Function Blocks (BFBs), used to specify general behaviour,
- Service Interface Function Blocks (SIFBs), used to interface the environment of a FB network, and
- Composite Function Blocks, emerging from a composition of BFBs/SIFBs and (inner) composite FBs.

In common all FB types provide an *interface* defining *input events* with associated *input* data variables, and *output events* with associated *output* variables.

The operation of a BFB is defined (in a finite state machine like manner) by its Execution Control Chart (ECC), input/output events, and input/output/local variables. A transition condition (edge in the ECC) is either a single *input* event, a Boolean expression (guard) on input/output and local variables, or a combination thereof. Each state in the ECC, implies an ordered set of zero or more algorithms to execute and *output* events to emit. On the arrival of an input event (provided by the resource scheduler), the associated input data connections are first sampled to the input variables, then transition conditions from the current state are checked (in order given by occurrence in the underlying XML). On a transition the algorithms of the target state are sequentially executed (implying potential output events and local and output variable updates), the input event is (conceptually) consumed, and further transition conditions (from the target state) inspected transitively until no more transitions are possible.

The operation and implementation of SIFBs are left undefined in the standard. CFBs provides a mere hierarchical abstraction and carry no functionality in their own right.

2) *Deployment*: For the deployment, the *application* is partitioned onto a set of *resources*, which in turn are mapped onto a set of *devices*. Communication crossing resource boundaries (e.g., inter device communication) must pass through SIFBs.

The standard also associates other properties to the notions of resources and devices, allowing *shielded* execution and *dynamic re-configuration*.

3) *Execution model*: The execution model is event driven (asynchronous), where each event may be associated with a set data connections. A *device* may provide one (or more) *resource(s)* responsible for the scheduling of events. The order of events delivery and execution is undefined, hence the execution model is non-deterministic.

B. Real-Time For the Masses

Real-Time For the Masses (RTFM) is a set of experimental languages and tools, designed to bring state-of-the-art embedded software technologies to the mainstream. The Model of Computation (MoC) has the outset of the Stack Resource Policy (SRP) [13], which brings numerous advantages to preemptive scheduling of systems with shared resources, e.g., deadlock free execution, bound priority inversion, as well as memory and CPU efficient implementation. Given Worst Case Execution Times (WCETs) for the tasks and their inner claims, schedulability test and response time analysis is readily available for single-core systems [13], as well as and multi-cores [14].

1) *RTFM, Model of Computation*: The notation from [13] has been adapted to our use in the context of RTFM. For this presentation we use task instance and task interchangeably: t_i , is a *task* (job) with run-to-completion semantics; r_j , defines a resource, a task may *claim* (request and release) resources in a LIFO nested manner; and e_i , is an event (job request) triggering the execution of the corresponding task t_i .

To the end of SRP based analysis and deadline monotonic (DM) scheduling we define: $dl(e_i)$, is the deadline for the execution of task t_i ; $p(t_i)$, is the static priority for t_i ; $ia(e_i)$, is the minimum inter-arrival time (or period) of e_i ; $c(t_i)$, is the worst case execution time for t_i ; and $rp(e_i)$, is the response time to the event e_i (i.e., task t_i).

In the context of RTFM, claiming a resource can be seen as entering a critical section. It extends the SRP task model with the ability for tasks to emit events (hence allowing asynchronous task chains to be expressed). In comparison to SRP, resources in RTFM are single unit.

2) *RTFM Run-Time Systems*: Models in RTFM be efficiently executed by the *RTFM-kernel* on single-core bare metal platforms [15] and by the *RTFM-RT* [16] onto hosted environments that provide threading primitives.

3) *RTFM-core*: The RTFM-core programming language [17], [10] builds on the RTFM MoC. From the input -core program the RTFM-core compiler generates C code output referring to the RTFM-kernel/RTFM-RT primitives for the scheduling. During compilation the task set is analyzed for inconsistency (cyclic resource claims are rejected). For valid models *resource ceilings* for SRP based scheduling are derived and provided as input for run-time scheduling (as implemented in the RTFM-kernel/RTFM-RT).

C. IEC 61499 mapping to the RTFM Model of Computation

In previous work, a mapping from device/resource level IEC 61499 models to the RTFM MoC (as provided by the RTFM-kernel API) has been proposed [18]. For this presentation, we adopt a similar approach, while mapping to the RTFM-core language instead of the RTFM-kernel scheduling primitives. This has the advantage that model level analysis and optimizations (performed by the RTFM-core compiler) can be put to use. (For detailed descriptions of the original work, we refer the reader to [18].)

III. RTFM TIMING SEMANTICS

A. Permissible Execution Window

Each *event* (e) is associated with an *baseline* $bl(e)$ (absolute point in time for the arrival of the event), a relative *deadline* $dl(e)$ (indicating the timing requirement), and a minimum inter-arrival time $ia(e)$. An event e_i , is associated (triggers) a corresponding task (instance) t_i (can be seen as a job request related to SRP).

The Permissible Execution Window (PEW) for (an instance of) t_i is the range in time from its baseline $bl(t_i) = bl(e_i)$ to its absolute deadline $dl(t_i) = bl(e_i) + dl(e_i)$, Figure 2.

A task t_i , may emit further synchronous and asynchronous events e_j , which by default inherits the sender's timing properties ($bl(e_j) = bl(e_i)$, $dl(e_j) = dl(e_i)$), and hence the corresponding task t_j executes under the sender's PEW. On emitting a synchronous event, the corresponding task is (directly) executed and the sender is suspended until t_j completes), while on emitting an asynchronous event the sender continues execution.

An asynchronous event e_j can be associated explicit timing properties in terms of a relative baseline offset $af(e_j)$ (such that $bl(e_j) = bl(e_i) + af(e_j)$) and/or an explicit $dl(e_j)$. We consider models valid only if the corresponding $dl(t_j) > bl(t_j)$ (the size of the PEW is > 0), and that $dl(e_j) \leq ia(e_j)$, which are common assumptions to real-time scheduling. In the following, we assume systems to be schedulable if not else stated.

B. The RTFM-core language

1) *Syntax*: For this paper we extend on the original RTFM-core language [10] by introducing the aforementioned time semantics. The `async` defines for the task instance (job request) a *permissible execution window* (absolute base- and dead-lines), inherited or computed from relative `after` and `before` timing information given in the -core program. The `pend` defines a new baseline from the current time and an optional deadline.

A simplified grammar is depicted in Figure 1. The `CCode` terminal denotes the presence of embedded C-code, occurring either at the top level, or inside a statement. (Func's are used merely to facilitate modularization.)

We discriminate between *Task/ISR definitions* given in the input program, and *Task/ISR instances* derived by analyzing the input program. During compilation, (for current run-time targets) static priorities are derived for each `ISR` (by deadline) and `Task` instance from the timing constraints of the originating asynchronous event.

C. Dynamic Semantics

In the following, we give an informal dynamic (run-time) semantics for the -core language, following the notions of *tasks* and *resources*.

Each `ISR`, `Task` and `Reset/Idle` definition is bound to a finite sequence of statements and is referred to as a *task* definition. Execution of a `ISR/Task` (a *task instance*) is triggered by the occurrence of a corresponding *event* and should run-to-completion (i.e terminate). An event can either

Top	::=	#> CCode <# ISR Id Before? { Stmt } Task Id { Stmt } Func Id (CCode) { Stmt } Reset { Stmt } Idle { Stmt } Top Top
Stmt	::=	#> CCode <# claim Id { Stmt } async After? Before? Id () pend Before? Id () sync Id (CCode) Stmt Stmt
After	::=	after Int
Before	::=	before Int

Figure 1. Simplified grammar for RTFM-core

occur from the environment (typically an interrupt) or be emitted internally. `Idle` is a (potentially non-terminating) process with the lowest priority in the system, invoked once by the underlying run-time system at startup (after run-to-end execution of the `Reset` task).

The `Tasks` are concurrent in our model (and may execute in parallel, besides `Reset` that run before any other task is released). Their associated priorities *may* be used by the underlying scheduler (and corresponding resource and scheduling analysis). During execution a *task* may request (`claim`) a (single-unit) *resource* for the duration of a *critical section*. Following the grammar, resources will be claimed in a LIFO manner. Functions execute synchronously (`sync`) on behalf of the sender.

The event e_{Reset} has a baseline set to the current device time at the point when `Reset` has run-to-end. In a distributed setting, device time may be synchronized and thus common to all devices. In our model, the `Reset` task executed as part of device/scheduler initialization, (not part of the run-time scheduling for the system), and thus has no notion of specific deadline.

D. Cyclic task chains

Let $d(t)$ denote the task *definition* of t . Assume a task chain $t_i \xrightarrow{async^*} t_k$. In case $d(t_i) = d(t_k)$, i.e., t_i and t_k refers to the same task definition, we have a cyclic task chain. The period p of the task chain (and all its corresponding sub-task) is given as:

$$p = \sum_{n=i}^k X_n \quad (1)$$

where X_n is the baseline offset (`async after` X_n).

1) *Well-formedness*: To ensure schedulability, each sub-task of a periodic task chain should have a deadline shorter or equal to its inter-arrival time ($dl(t_n) \leq p$). Moreover, to ensure bound number of task instances, task chains must originate from the `Reset` task and each $d(t)$ may be referred to by at most one cyclic chain.

E. Examples

For brevity, we exclude the execution of `Reset` in the timing diagram. Device time is assumed 0 at the point when `Reset` has run-to-end, i.e., $bl(e_{Reset}) = 0$.

1) *Example 1:* In our first example we demonstrate a simple delay. The task t_1 is triggered with $bl(t_1) = 0s$ inherited from the `Reset` and the absolute deadline $dl(t_1) = 500ms$ derived from the relative timing constraint (line 11, Listing 1). Task $t_2()$ is triggered with arrival $bl(t_2) = 1s$ and absolute deadline $dl(t_2) = 2s$ by the `async` event emitted from task t_1 (line 3, Listing 1).

```

1 Task t1() {
2   #>printf("t1\n");<#
3   async after 1s before 1s t2();
4 }
5
6 Task t2() {
7   #>printf("t2\n");<#
8 }
9
10 Reset {
11   async before 500 ms t1();
12 }

```

Listing 1. Ex1.core simple task chain and message passing.

Figure 2. Timing for Example 1.

2) *Example 2:* Our second example defines a system, that on reset triggers the periodic task t_1 (with $bl(t_1) = n, dl(t_1) = n + 1s$), where n increments by 1 for each task instance/invocation (i.e., a period of 1s).

```

1 Task t1() {
2   #>printf("t1_\n");<#
3   async after 1s before 1s t1();
4 }
5
6 Reset {
7   async t1() before 1s;
8 }

```

Listing 2. Ex2.core periodic task.

Figure 3. Timing for Example 2.

IV. TIMING SEMANTICS FOR IEC 61499

In this section we propose how the aforementioned real-time semantics (Section III) can be adopted in the IEC 61499 standard. Our approach builds on, and extends, the mapping of the IEC 61499 models to the RTFM MoC [18] (briefly reviewed in Section II-C).

A. Mapping of design elements

Event properties: The definition of events follows that of the RTFM MoC, and the syntax (Figure 4).

```

Event ::= async After? Before? Min?
        | pend Before? Min?
        | sync

After  ::= after Int

Before  ::= before Int

Min     ::= min Int

```

Figure 4. RTFM event grammar for IEC 61499.

Event source: An event source, injects events from the environment into the FB design. In IEC 61499, event sources are defined using SIFBs. We propose RTFM-core as a (possible) specification language for SIFBs. For bare metal implementation at device level, event sources are implemented as ISRs (interrupt service routines), and the generated code automatically infers the baseline for the event/task. If choosing to implement SIFBs outside the RTFM MoC, the source event should be associated a baseline using the `pend` option (which assigns the baseline to the current time). For the analyses the `min` option should be used to define the minimum inter-arrival time for the event source. A SIFB output event that occurs on behalf of conveying an IEC 61499 event e should propagate the timing properties of e (i.e., PEW and inter-arrival).

Event chains: For (asynchronous) events we have the option inherit or explicitly state timing constraints, (Figure 4). The `after` option allows us to define complex timing patterns (for which delays and periodic behavior are trivial cases), without the need to infer special design elements. Notice, the baseline offsets are statically assigned, which allows for compile time analysis on the well-formedness of the design as well as offline schedulability analysis.

Point of association: The IEC 61499 standard implies FBs as being loosely coupled components/elements for application design. From the component point of view, one could argue that event properties should therefore be stated as part of the FBs. This was also the outset for the Real-Time Execution of IEC 61499 [4], as well as our previous work [18]. However, this approach is somewhat limiting, as output events may be split. Associating properties to each connection (at network level) increases the flexibility, allowing downstream event handling under different timing constraints (similar to [3]).

In order to satisfy both the component view and allow for maximum flexibility we propose that *default* event properties should be stated as part of the FB event outputs, while allowing to be over-ridden by properties given for the connections at network level.

RTFM Tasks: Each chain of synchronous events is a *task* t_i triggered by an asynchronous event e_i . If events are merged (i.e., several output events trigger the same input event), we see this as distinct task chains (executing concurrently under the timing constraints of the specific triggering event).

RTFM Resources: For the handling of resources, we adopt the mapping developed in [18].

B. System model example

IEC 61499 System model: Figure 5, depicts an IEC 61499 model developed in the 4DIAC IDE [19]. The SIFB instances Ea1 and Ec1 capture the external events from the underlying platform (or platforms if deployed onto different devices) and trigger the execution of actions associated to a1.i1 and c1.i1 respectively. Figure 6 depicts the ECC of b1, showing the associated actions (algorithms) taskb1 for input event i1 and taskb3 for input event i3 respectively. The m is an FB (from the standard library) that merges (and serialize) incoming events (and associated data).

Figure 5. Example IEC 61499 system model.

Figure 6. ECC of FB b1.

Example: Event source: Assuming we deploy the system (application) onto a single device. The output events from SIFBs Ea1 and Ec1 are obvious event sources (to the IEC 61499 network). In order to define the baseline, and should thus either be implemented natively in RTFM-core (and there originating from some ISR of the underlying hardware), or emitted with the `pend` option. In either case for the analysis the minimum inter-arrival should be stated using the `min` option. Figure 7, depicts the timing property of the event Ea1.o1 $\rightarrow$ a1.i1 at the network level.

Figure 7. Timing property of Ea1.o1 $\rightarrow$ a1.i1 at network level.

Example: Event chains: If not explicitly stated, events are considered to be synchronous, i.e. the event chains originating from event sources will be treated as synchronous task chains in the model and for execution. For the example, this amounts to the set of tasks and resources depicted in Figure 8.

Example: Resources: The assumption that $dl(t) \leq ia(t)$ (and that the system is schedulable), *only* the RTFM resources that are shared between tasks chains triggered by *different* event sources needs to be protected. For the example, this gives us the (reduced) set of protected resources $r(m)$ (for the ECC of m), its data output $r(o)$ (for the connection m.OUT_1 $\rightarrow$ b1.di1) and the $r(b)$ (for the ECC of b1). Notice here, that the ECC of b1 triggers an output event o1 *only* on a incoming i3 event, which in turn occurs on behalf of the single source event Ec1.o1, thus neither the state of b2 nor the data connection b1.d $\rightarrow$ b2.d1 needs protection.

Figure 8. Example RTFM system model. We have the (external) events e_{a1} and e_{c1} , triggering the task chains headed by a_1 and c_1 respectively. Asynchronous events are marked as dashed, tasks are denoted by circles, boxes $r(m)$, $r(b)$ are resources for the corresponding FBs, while $r(o)$ is a resource for a shared connection.

Example: Scheduling: At device level, RTFM models can be scheduled efficiently by the RTFM-kernel, exploiting the underlying interrupt hardware[15]. Each synchronous task chain, amounts to a RTFM task. For the example, the task chain triggered by e_{a1} (with the sub-tasks $a_1 : m_1 : b_1$) and task chain triggered by e_{c1} (with the sub-tasks $(c_1) : a_2 : m_2 : b_1$, $(c_1) : b_3 : b_2$ and $(c_1) : c_2$). Notice, this gives an upper bound to the set of sub-tasks executed on behalf of occurred source events e_{a1}/e_{c1} . For a given configuration (ECC specifications of the FBs and FB states), actual execution involves a proper subset of the sub-tasks of the executing task.

Example: Analysis: Given Worst Case Execution Time (WCET) bounds for all sub-tasks, SRP based response time analysis can be readily applied. For a detailed description, see e.g., the original work of Baker [13]. In the following $c(t_i) = C_{t_i}$ denotes the WCET C of a sub-task t_i .

Example: CPU utilisation: Assuming the WCETs given in Figure 8. The total WCETs of the task (chains) triggered by $e_{a1} = 50a_1 + 1m_1 + 1b_1 = 52ms$, and $e_{c1} = 5c_1 + 5a_2 + 1m_2 + 1b_1 + 5b_3 + 2b_2 + 3c_2 = 22ms$ respectively. Assume $ia(a_1) = 1s$, $ia(c_1) = 60ms$, CPU utilization is $52/100 + 22/60 = 88.7/100$, hence the system *may* be schedulable, (as total CPU utilization is less than 1).

Example: Response time: Assume that $dl(e_{a1}) = 100ms$, and $dl(e_{c1}) = 30$ respectively. During compilation, static priorities are assigned (monotonically) based on decreasing deadlines, hence an assignment $p(e_{a1}) = 1, p(e_{c1}) = 2$ meets this requirement (a higher value indicates a higher priority).

The response time of a task t is under SRP computed as the sum of the WCET, interference by higher priority tasks, and blocking by lower priority tasks (known as priority

inversion). Under SRP blocking of a task t_i is limited to a single section where a lower priority task holds a resource, with a priority ceiling ($\lceil r \rceil$) larger or equal to $p(t_i)$. The priority ceiling of a resource r is computed as the maximum priority for any task that may access r . For our example $\lceil r(m) \rceil = \lceil r(o) \rceil = \lceil r(b) \rceil = 2$, since resources may be accessed by tasks at priority 2 for these (shared) resources.

Preemption time can be safely estimated by the interference of all tasks occurrences until its deadline. In our example $dl(e_{a_1}) = 100ms$, and $ia(e_{c_1}) = 60ms$, which implies that (in worst case) the execution of e_{a_1} will be preempted (at most) twice. This gives us a response time $rp(e_{c_1}) = c(e_{a_1}) + 2 * c(e_{a_2}) = 52 + 2 * 22 = 96ms$ (no blocking since no lower priority task exists). Hence, we satisfy the deadline requirement for $e_{a_1} = 100ms$.

Let us now assess the response time for e_{c_1} . Since it is the highest priority task in the system, interference (preemption) will not occur. However, it may be blocked by a lower priority task. In the example the worst case is when resource m , is held by e_{a_1} for the section (sub-tasks) $m1 : b1$ (notice, the sequential execution of b_1 on behalf of m_1). This gives us a blocking time of $c(m_1) + c(b_1) = 1 + 1 = 2$, and a total response time $rp(e_{c_1}) = c(e_{c_1}) = 22 + 2ms$. Hence, we satisfy the deadline requirement for $e_{c_1} = 30ms$.

Example: Conclusion: The given example demonstrate the device level mapping from IEC 61499 models to tasks and resources of the RTFM MoC, and showcase the use of the proposed timing semantics, both to specify timing requirements, to generate an outset for device level SRP based scheduling, and the relation to established methods for response time analysis.

V. CONCLUSIONS

In this paper we have proposed a real-time semantics, Real-Time-4-FUN, for the IEC 61499 standard. Events are associated with base- and deadlines, providing an outset for reasoning on timely properties at the model level. As allowing for non-deterministic execution, correctness is defined by the execution semantics, not as currently on basis of timing properties emerging from a specific deployment. This provides for a clear separation of concerns, where the designer can focus on properties of the application prior to, and separately from, deployment verification.

The proposed semantics is compositional, where baseline offsets provide a means to express complex (yet statically analyzable) timing patterns directly at model level (thus superseding the need for the implementation dependent timing components currently required using IEC 61499). Timing requirements are transitional and propagate to downstream sub-systems, (allowing the re-use of legacy IEC 61499 components), and can be utilized for scheduling both at device and network level. In this paper we have shown that, based on a translation to RTFM-tasks and resources, IEC 61499 models can be analyzed, deployed and verified. As a proof of concept the timing semantics has been experimentally implemented in the RTFM-core language and the accompanying (thread based) RTFM-RT run-time system.

ACKNOWLEDGEMENTS

The authors like to thank Dr. Alois Zotl (Fortiss) for fruitful discussions and feedback.

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology), and the EU ARTEMIS JU funding, within project ARTEMIS/0001/2013, JU grant nr. 621429 (EMC2) and VINNOVA (Swedish Governmental Agency for Innovation Systems) and Svenska Kraftnät (Swedish national grid).

REFERENCES

- [1] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 7, pp. 1165–1178, July 2008.
- [2] A. David, K. G. Larsen, A. Legay, U. Nyman, L.-M. Traonouez, and A. Wąsowski, "Real-time specifications," *International Journal on Software Tools for Technology Transfer*, pp. 1–29, 2013.
- [3] C. Schwab, M. Tangermann, A. Luder, A. Kalogeras, and L. Ferrarini, "Mapping of IEC 61499 function blocks to automation protocols within the TORERO approach," in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*, June 2004, pp. 149–154.
- [4] A. Zotl, *Real-Time Execution for IEC 61499*. Durham, North Carolina: International Society of Automation, 2008.
- [5] C. Pang, J. YAN, and V. Vyatkin, "Time-complemented event-driven architecture for distributed automation systems," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.
- [6] J. Provost, J.-M. Roussel, and J.-M. Faure, "A formal semantics for Grafcet specifications," in *7th IEEE Conference on Automation Science and Engineering (IEEE CASE 2011)*, Trieste, Italy, Aug. 2011, pp. 488–494.
- [7] N. Bauer, R. Huuck, B. Lukoschus, and S. Engell, "A unifying semantics for sequential function charts," in *SoftSpec Final Report*, ser. Lecture Notes in Computer Science, H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, Eds., vol. 3147. Springer, 2004, pp. 400–418.
- [8] The Timber Language. (webpage) Last accessed 2011-04-15. [Online]. Available: <http://www.timber-lang.org>
- [9] J. Eriksson, S. Aittamaa, J. Wiklander, P. Pietrzak, and P. Lindgren, "SRP-DM Scheduling of Component-Based Embedded Real-Time Software," *First International Workshop on Dependable and Secure Industrial and Embedded Systems*, 2011.
- [10] P. Lindgren, M. Lindner, A. Lindner, D. Pereira, and L. M. Pinho, "RTFM-core: Language and Implementation," in *Accepted to: 10th IEEE Conference on Industrial Electronics and Applications (ICIEA 2015)*, 2015.
- [11] IEC standards. (2014, Mar.) International Electrotechnical Commission. [Online]. Available: <http://www.iec.ch/>
- [12] J. Christensen, T. Strasser, A. Valentini, V. Vyatkin, A. Zotl, J. Chouinard, H. Mayer, and A. Kopitar, "The IEC 61499 Function Block Standard: Software Tools and Runtime Platforms," in *ISA Automation Week*, 2012.
- [13] T. Baker, "A stack-based resource allocation policy for realtime processes," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*, Dec. 1990, pp. 191–200.
- [14] A. Burns and A. Wellings, "A schedulability compatible multiprocessor resource sharing protocol – mrsp," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, July 2013, pp. 282–291.
- [15] J. Eriksson, F. Haggstrom, S. Aittamaa, A. Kruglyak, and P. Lindgren, "Real-time for the masses, step 1: Programming API and static priority SRP kernel primitives," in *SIES. IEEE*, 2013, pp. 110–113.
- [16] M. Lindner, A. Lindner, and P. Lindgren, "," in *RTFM-RT: a threaded runtime for RTFM-core to execute IEC 61499 models*, Sept 2015.
- [17] RTFM: Real-Time For the Masses. (webpage) Last accessed 2015-02-17. [Online]. Available: <http://www.rtfm-lang.org>
- [18] P. Lindgren, M. Lindner, V. Vyatkin, and et. al., "Real-Time Execution of Function Blocks for Internet of Things using the RTFM-kernel," in *Proceedings of 2014 IEEE 19th International Conference on Emerging Technologies & Factory Automation (ETFA 2014)*, 2014.
- [19] 4DIAC - Organization. (webpage) Last accessed 2015-02-17. [Online]. Available: <http://www.fordiac.org>