TIERED DURATIONS: SCHEDULING AT DIFFERING TIME RESOLUTIONS

Eike Schulte, Stephan Balduin
OFFIS – Institute for Information Technology
email: eike.schulte@offis.de

KEYWORDS
Co-simulation, Scheduling, Discrete Simulation, Simulation interfaces, Event-oriented

ABSTRACT

Co-simulations of cyber-physical energy systems often comprise components running at vastly different time scales, ranging from nano-second communication within a tightly coupled system to 15 minutes or longer for, e. g., power-flow calculations. We introduce *tiered durations*, which allow coupling of such components without requiring the slower ones to be aware of the high-frequency steps of the faster ones while maintaining clear semantics for the simulation as a whole, extending earlier concepts of super-dense time and same-time loops. On top of this, *simulator groups* allow creating consistent data-flow graphs using tiered durations. Both tiered durations and simulator groups have been implemented in our co-simulation framework MOSAIK.

INTRODUCTION

Simulations of Cyber-Physical Energy Systems (CPES) often deal with vastly different time scales within a single simulation. For example, the power grid might be simulated in 15-minute durations (as that is the frequency of many energy markets), whereas agent-based models connected to it communicate at sub-second speed. While it would be possible (and more correct, even), to simulate the agent's real communication times, this is often quite cumbersome without promising more accurate results. We would, therefore, like to abstract away the precise timing of these communications while preserving their logical ordering. The following example will serve as motivation and for illustration purposes throughout the paper:

Example. A fleet of battery cells are connected to a joint inverter and are controlled by a software controller. Each of the cells, the controller, the inverter, and the power grid are represented as separate components in the simulation. The power grid simulation runs at 15-minute durations. At the beginning t of each duration, the battery cells and controller negotiate the use of the inverter based on the state of the grid in the previous duration and some internal state of the inverter. This results in an indeterminate number of steps, which should all be considered to be happening at time t but are still ordered. Once the negotiation is over, the cells relay their DC power values to the inverter, which then sends its real and reactive power to the grid. Figure 1 illustrates the simulators and their connections.

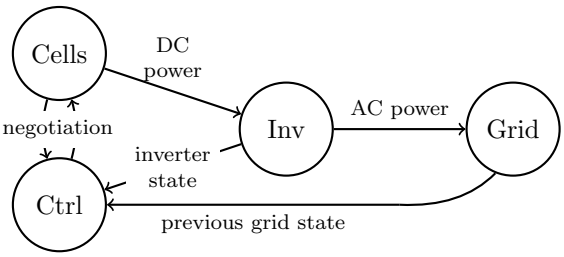

Figure 1: The example scenario

To support scenarios like this, co-simulation frameworks often provide *super-dense time* or *same-time loops*, allowing the different time scales to be separated.

For example, in version 3.0 the co-simulation framework MOSAIK was augmented with support for event-based scheduling and *same-time loops* (Ofenloch et al. 2022), which resolve this problem for the case of two simulators negotiating without advancing time. However, the approach becomes intractable as the number of negotiating simulators increases, as it subtly depends on the order in which simulators are started and connected.

Another popular approach is *super-dense time* where time is augmented by a second component explicitly ordering the simultaneous events (e. g. Eker et al. (2003), Blochwitz et al. (2011)). We will call this component the *sub-step*. However, when applying this approach globally, assigning an a-priori stepping time to a component like the power grid simulator in the example is impossible. After all, it is supposed to run after the negotiation is over, which takes an indeterminate number of steps, so the sub-step of the power grid is only known after the negotiation. At the same time, the power grid simulator should be able to set its own next step; it should not be necessary to trigger it from the inverter.

In this paper, we present an extension of the super-dense time approach, which we call *tiered time* and which we have implemented in version 3.3 of MOSAIK. The extension is two-fold. First, we allow our times to have an arbitrary number of components (instead of just two), and second, we allow different simulators to work

at different time resolutions, i. e., with times having different numbers of components. This allows us to avoid assigning a sub-step to a component like the power grid simulator where it is not needed. The main challenge then lies in how to relate times of simulators running at different time resolutions, which is resolved by the introduction of *tiered durations*.

The organization of this paper is as follows: This paragraph is followed by a short overview of related work. In the next section, we briefly sketch the principles behind MOSAIK's time synchronization. Then, we introduce tiered times and tiered durations, which are used to represent the points in time where steps occur and the durations between those times, and explain their basic properties. This is followed by an explanation of simulator groups, which provide the interface a scenario author uses to take advantage of tiered times. We conclude our work in the last section.

Other related work

Super-dense time has been discussed for some time, apparently first by Maler et al. (1992). Usually, a "standard" representation of time (i. e. the real numbers or the integers, depending on whether or not time is considered discrete) is augmented by a second natural-number component, ordering events that happen simultaneously (e. g. Broman et al. (2015).

In *Hybrid simulation: it's about time*, Cremona et al. (2019) discuss the super-dense time in the context of the FMI standard (Blochwitz et al. 2011), together with a negotiation mechanism for a shared time resolution between simulators.

In *Toward a Theory of Superdense Time in Simulation Models*, Nutaro (2020) gives an axiomatization of what is required for a "time" in the context of discrete event systems (e. g. Zeigler et al. (2019)), along with several examples of systems of super-dense time fulfilling those axioms. The axiomatization presented there does not allow different simulators to use different time resolutions, though, and thus cannot be applied to our tiered durations.

TIME SYNCHRONIZATION IN MOSAIK

We will first briefly explain MOSAIK's scheduling algorithm for normal, non-tiered time, in which case stepping times are always integers. For a more in-depth explanation, we refer to MOSAIK's documentation[1].

MOSAIK uses a global conservative scheduler that steps each simulator independently, meaning that two simulators running simultaneously do not need to agree on the current time (except when running in real-time mode, which is beyond the scope of this paper). Ordering between the steps of two different simulators is purely based on the connections established between them by the user in their scenario script. For example, when an output of simulator A is connected to an input of simulator B, and both simulators have a step scheduled at the same time t, MOSAIK will step simulator A first so that its output is available for simulator B's step. This is provided that the connection is not marked as time-shifted. More generally, if a time shift of u is specified for that connection, B will be allowed to step ahead of A, in the sense that its step for time $t+u$ can be performed as soon as A has passed the time t, and B will use A's newest output from that time. This allows loops between two or more simulators to be resolved explicitly.

While the time shifts between simulators are static and known to MOSAIK, the precise stepping times are not, as they can be chosen by the simulators during the simulation and can also depend on data sent between simulators. To perform its scheduling, MOSAIK keeps track of both the already-determined future steps of each simulator and of each simulator's *progress*, which is its earliest potential future stepping time, based on the scheduled steps of all simulators and the given time-shifts. The time of the earliest scheduled step will only go down as earlier steps are scheduled (until a step is actually performed), while the progress can only increase (whenever simulators decide not to produce events that could have otherwise triggered earlier steps). Once the time of the earliest scheduled step and the progress are the same, that step is committed as the next step to be performed. MOSAIK then waits until all simulators that could provide data for this step have progressed past this point before stepping the simulator with the collected inputs.

Depending on the simulators' connections and stepping behavior, some might step ahead of others. This doesn't affect correctness, provided the simulators do not communicate "behind MOSAIK's back".

TIERED TIME AND TIERED DURATIONS

As seen in the introduction, integer time steps are often impractical when integrating communication into co-simulation scenarios, as communication happens at a much faster time scale than the rest of the simulation. Fine-tuning the stepping times of all components to support this is frequently cumbersome, especially when communication delays are not the focus of the investigation.

Tiered times are a solution to this. Instead of assigning each step a single integer as its time, tuples $(t_0, \ldots, t_{l-1})$ of one or more non-negative integers are used. Each component of these tuples is called a *tier*, and the number l of tiers is called the tiered times' *length*. Tiered times are ordered lexicographically, which allows any step to be subdivided into infinitely many smaller steps by going one tier down. All

[1] `https://mosaik.readthedocs.io/en/latest/scheduler.html`

tiered times associated to one simulator will have the same length, which we call the simulator's *(time) resolution*. Crucially, however, different simulators possibly use different time resolutions.

To relate tiered times between simulators, we introduce *tiered durations*, which represent lengths of time. They act on tiered times, in the sense that it is possible to add a tiered duration u to a tiered time t to get tiered time $t + u$. Three basic types of tiered durations are necessary:

(A) *Time shift*: Add a non-negative integer to each component of t to receive a tiered time of the same length.
(B) *Truncation*: Discard the last tier of t (if it has length ≥ 2). This represents the idea that when a simulator waits for another simulator running at a higher resolution, all sub-steps at the higher resolution should happen "at once".
(C) *Extension*: Append a tier with value 0 at the end of t. This represents the same idea as above, except that now the higher-resolution simulator waits for the lower-resolution one.

While scheduling, it is often necessary to add several of these basic tiered durations to a tiered time one after the other. It, therefore, turns out to be very useful to be able to add two tiered durations as well, resulting in a combined tiered duration that has the same effect. More formulaically, we want an operation on tiered durations, also written $+$, such that for any tiered time t and tiered durations u, v, we have

$$(t + u) + v = t + (u + v), \tag{1}$$

so that it does not matter whether we add the tiered durations to the tiered time one after the other or combine them first.

However, if u and v are among the basic types (A) to (C) listed above, the tiered duration $u + v$ might no longer be expressible as one of these basic tiered durations. This forces us to choose a more general definition. We will also associate a *pre-length* to each tiered duration, which is the length of the tiered times and tiered durations to which it can be added. This serves mostly as a sanity check, preventing mix-ups between different tiered durations or the order in which they are added.

Definition. A *tiered duration* u of *pre-length* k and *length* l (also a (k, l)*-tiered duration* for short) is an l-tuple $(u_0, \ldots, u_{l-1})$ of non-negative integers together with a *cut-off length* $\mathrm{co}(u)$, where we require that $1 \leq \mathrm{co}(u) \leq \min\{k, l\}$.

The sub-tuple $(u_0, \ldots, u_{\mathrm{co}(u)-1})$ is called the *addition part* u_{add} of the tiered duration, the sub-tuple $(u_{\mathrm{co}(u)}, \ldots, u_{l-1})$ is the *replacement part* u_{repl}. We also write the tiered duration as $(u_{\mathrm{add}} \mid u_{\mathrm{repl}})$ or $(u_0, \ldots, u_{\mathrm{co}(u)-1} \mid u_{\mathrm{co}(u)}, \ldots, u_{l-1})$.

We denote by 0_k the (k, k)-tiered duration $(0, \ldots, 0 \mid)$ with k zeros.

Addition of a (k, l)-tiered duration u and an (l, m)-tiered duration v results in a (k, m)-tiered duration defined by

$$(u + v)_i = [i < \mathrm{co}(v)]u_i + v_i \qquad \text{for } 0 \leq i < m, \tag{2}$$
$$\mathrm{co}(u + v) = \min\{\mathrm{co}(u), \mathrm{co}(v)\}, \tag{3}$$

where the square brackets in (2) are Iverson brackets, evaluating to 1 if the condition inside is true, and evaluating to 0 otherwise. (This 0 should be considered a "strong zero", i. e. despite u_i technically being undefined if $i \geq l$, we consider $[i < \mathrm{co}(v)]u_i = 0$ in this case, as $i \geq l \geq \min\{l, m\} \geq \mathrm{co}(v)$.)

Adding an (l, m)-tiered duration to a tiered time of length l works similarly, following equation (2) with u as the tiered time, v as the tiered duration.

In essence, when adding a tiered duration to a tiered time or duration, the addition part is added to the corresponding tiers of the first summand, while the replacement part replaces the remaining tiers.

Proposition. *The addition of tiered durations is associative, and the action on tiered times fulfills* (1)*. For each length l, the tiered duration 0_l is a neutral element when added to (k, l)-tiered durations from the right and when added to (l, k)-tiered durations from the left.*

Proof. Straightforward calculation using (2), (3). □

Note. The addition of tiered durations is not commutative. We apologize for denoting it using $+$, regardless.

A basic time shifts is represented by $(u_0, \ldots, u_{k-1} \mid)$ where $u_0, \ldots, u_{k-1}$ are the components to be added. A truncation is given by $(0, \ldots 0 \mid)$ (with as many zeros as tiers should remain), and an extension is given by $(0, \ldots, 0 \mid 0)$ (with as many zeros total as the destination simulator's time resolution).

During scenario setup, the "shortest" connection between two simulators will be needed occasionally, which necessitates comparing tiered durations. Denoting by $\leq$ the to-be-defined order on tiered durations (and using the same notation for the lexicographic order on tiered times), we should expect that for any two tiered times s and t with $s \leq t$ and tiered durations u and v with $u \leq v$, it follows that $s + u \leq t + v$. This necessarily means that some tiered durations won't be comparable.

This effect can already be witnessed when the tiered time summands are the same. For example, consider the following addition table:

	$+$	u $(0, 0 \mid)$	v $(0 \mid 1)$
t^0	$(0, 0)$	$(0, 0)$	$(0, 1)$
t^1	$(0, 2)$	$(0, 2)$	$(0, 1)$

In the row for t^0, the result from adding v is bigger, whereas in the row for t^1, the result from adding u is bigger. This means that the two tiered durations u and v cannot be ordered, and a partial order will have to do.

To define this order, we introduce a bit of notation: For a tiered duration u of length l and $0 \leq i \leq j \leq l$, denote by $u_{i:j}$ the tuple $(u_i, \ldots, u_{j-1})$. In particular, if u has length l, $u_{0:l}$ is the tuple consisting of the tiers of u, but the cut-off has been "forgotten". We also use this notation for tiered times.

Definition. Let u and v be two (k,l)-tiered durations and let $c = \min\{\mathrm{co}(u), \mathrm{co}(v)\}$. Then we define $u \leq v$ if $u_{0:c} < v_{0:c}$ or if $u_{0:l} \leq v_{0:l}$ and $\mathrm{co}(u) \leq \mathrm{co}(v)$, where the tuples are compared lexicographically in both cases.

Proposition. *Let s, t be tiered times of length k and let u, v be (k,l)-tiered durations. If $s \leq t$ and $u \leq v$, then*

$$s + u \leq t + v. \tag{4}$$

Proof. We will repeatedly use the fact that (4) holds if s, t, u, and v are tuples; $+$ denotes the usual componentwise addition; and $\leq$ is the lexicographic order. We label this version (4)′. Further, this inequality is strict if and only if at least one of the inequalities between s and t or u and v is also strict.

Also, for any tuples x and y of length l and any splitting point i, the lexicographic inequality $x \leq y$ holds if and only if $x_{0:i} < y_{0:i}$ or both $x_{0:i} = y_{0:i}$ and $x_{i:l} \leq y_{i:l}$. For this proof, we call this the *splitting property*.

Let $c = \min\{\mathrm{co}(u), \mathrm{co}(v)\}$ as in the definition. We have to consider two cases:

Case $u_{0:c} < v_{0:c}$. We only need to look at the addition parts, where $(s+u)_{0:c} = s_{0:c} + u_{0:c}$ and $(t+v)_{0:c} = t_{0:c} + v_{0:c}$. Hence, $(s+u)_{0:c} < (t+v)_{0:c}$ by (4)′, and going to the full tiered times $s+u$ and $t+v$ cannot reverse this.

Case $u_{0:l} \leq v_{0:l}$ and $\mathrm{co}(u) \leq \mathrm{co}(v)$. Then c is $\mathrm{co}(u)$ and we let $d = \mathrm{co}(v)$. We consider the pure addition part $0:c$, the mixed part $c:d$, and the pure replacement part $d:l$ in order. We have $(s+u)_{0:c} = s_{0:c} + u_{0:c} \leq t_{0:c} + v_{0:c} = (t+v)_{0:c}$ by (4)′ and if the inequality is strict, we are done. Otherwise, we may conclude $s_{c:d} \leq t_{c:d}$ and $u_{c:d} \leq v_{c:d}$ from the splitting property. Further, $(s+u)_{c:d} = u_{c:d}$, but $(t+v)_{c:d} = t_{c:d} + v_{c:d}$. As $t_{c:d}$ consists of non-negative components by the definition of tiered times, $(s+u)_{c:d} \leq (t+v)_{c:d}$. Again, if the inequality is strict, we are done, and otherwise, $u_{d:l} \leq v_{d:l}$. But since we are now in the replacement part, $u_{d:l} = (s+u)_{d:l}$ and $v_{d:l} = (t+v)_{d:l}$, so we are done. □

Because the ordering is not total, not every set of (k,l)-tiered durations has a minimum. However, two tiered durations with the same cut-off length will always be comparable, as in that case, the definition reduces to the lexicographic order, which is total. Therefore, a set of (k,l)-tiered durations has at most $\min\{k,l\}$ minimal elements, making storing all minimal elements feasible whenever a minimum does not exist. In practice, a minimum exists for most simulator pairs.

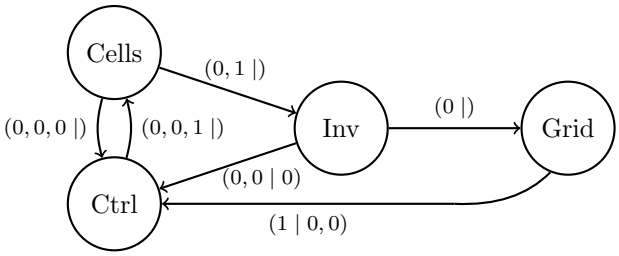

Figure 2: The example scenario with added time shifts

Integration into mosaik's scheduling

The mechanism described in the section *Time synchronization in mosaik* works essentially unchanged for tiered times and tiered durations. Each simulator's steps now happen at tiered times (of length corresponding to that simulator's time resolution). The time shifts on connections between simulators are given by (k,l)-tiered durations where k is the source simulator's time resolution, and l is the destination simulator's time resolution. (Because the involved simulators may have different time resolutions that need to be lined up, even non-time-shifted connections have tiered durations assigned, with all tiers 0 and appropriate cut-off lengths.)

Example. In our example, both the battery cells and the controller get a time resolution of 3, the inverter gets a time resolution of 2, and the power grid simulation runs at time resolution 1. See Figure 2 for the time shifts. At any main step t, the inverter and the cells run first and send data to the controller. The inverter schedules a step for time $(t,1)$. The controller exchanges messages with the cells for the negotiation, during which their tiered times are of the form $(t,0,s)$. As $(t,0,s) + (0,1\ |) = (t,1)$, which is not past the inverter's scheduled step, the inverter will wait. Once the negotiation ends, the cells' progress will increase to $(t,1,0)$, at which point the inverter can run again because $(t,1,0) + (0,1\ |) = (t,2)$ now has passed its scheduled step. It sends data to the grid and schedules its next step at time $(t+1,0)$. This will allow the grid to perform its step at time (t).

SIMULATOR GROUPS

MOSAIK's tiered-time system is not exposed to the user directly. Instead, users create *groups* in their scenario script, which can be nested, and start their simulators within them. At the base level, simulators use tiered time with one tier. Each group adds one extra tier to the resolution of simulators contained within.

Each connection between simulators is associated with a tiered duration. When a connection leaves a group, a tiered duration of type (B) is added; when a group is entered by the connection, a tiered duration of type (C) is added. When the user specifies an (integer) time shift, this is represented by a tiered duration of

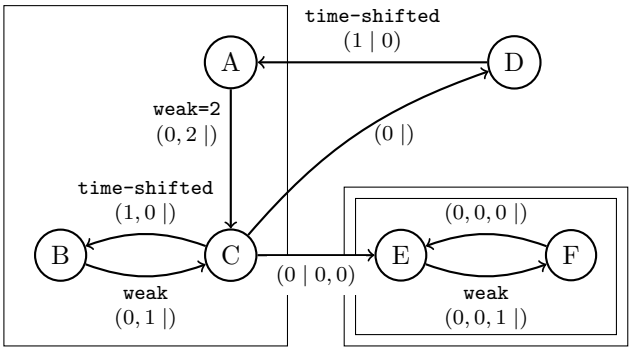

Figure 3: Example of a simulation setup. Rectangles denote simulator groups.

type (A) with the given time shift placed in the zeroth tier. Additionally, the user can specify a *weak* shift; this is also represented as a type (A) tiered duration, but placing the shift in the tier corresponding to the innermost group shared by both simulators. It is an error to specify a weak shift if the simulators share no group.

Example. Figure 3 shows a setup of six simulators illustrating these rules. The weak connection between B and C leads to a non-zero entry in the first tier (counting tiers from zero), while the weak connection from E to F materializes in the second tier. A weak connection from C to E would not be allowed because these simulators do not share a group.

It also illustrates a case where the non-totality of our order becomes relevant. Namely, simulator C loops to itself in two ways: via simulator B, with a total duration of $(1, 0\,|) + (0, 1\,|) = (1, 1\,|)$ and via simulators D and A, with a total duration of $(0\,|) + (1 \mid 0) + (0, 2\,|) = (1 \mid 2)$. As these are not comparable, both need to be considered when determining how soon output from C can influence simulator C's own input.

API considerations

As MOSAIK already had a system of weak connections in version 3.2 and earlier, it was important to design the new API to avoid existing simulations suddenly producing different results without notice. This motivated introducing groups and making their use mandatory when using weakly shifted connections. This way, users without weak connections are completely unaffected, while users with existing weak connections get an error message referring them to the appropriate documentation.

For very simple setups involving just one weakly connected pair of simulators, the error is resolved by simply placing both simulators in the same group. For more complicated setups of weak connections, some decisions need to be made on distributing them into groups, which depend on the intended stepping and data-flow behavior. In these cases, the old system would call the simulators in an essentially random fashion.

CONCLUSIONS

By introducing tiered durations and simulator groups into MOSAIK's scheduling algorithm, stepping more than two simulators without advancing the main time has become possible, while these setups have been difficult or impossible to achieve previously. This paves the way for simulations integrating components operating at vastly different time scales.

In a future paper, we plan to give an axiomatization of tiered times and durations in a similar to the one in Nutaro (2020).

ACKNOWLEGDEMENTS

This research is a part of project ReCoDE, funded by the German Federal Ministry for Economic Affairs and Climate Action (FKZ 03EI6093A).

REFERENCES

Blochwitz T.; Otter M.; Arnold M.; Bausch C.; Clauss C.; Elmqvist H.; Junghanns A.; Mauss J.; Monteiro M.; Neidhold T.; Neumerkel D.; Olsson H.; Peetz J.V.; and Wolf S., 2011. *The Functional Mockup Interface for Tool independent Exchange of Simulation Models.* In *Proceedings of the 8th International Modelica Conference*. 105–114. doi:10.3384/ecp11063105.

Broman D.; Greenberg L.; Lee E.A.; Masin M.; Tripakis S.; and Wetter M., 2015. *Requirements for hybrid cosimulation standards.* In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. Association for Computing Machinery, New York, NY, USA, HSCC '15, 179–188. doi:10.1145/2728606.2728629.

Cremona F.; Lohstroh M.; Broman D.; Lee E.A.; Masin M.; and Tripakis S., 2019. *Hybrid co-simulation: it's about time. Software & Systems Modeling*, 18, no. 3, 1655–1679. doi:10.1007/s10270-017-0633-6.

Eker J.; Janneck J.; Lee E.; Jie Liu; Xiaojun Liu; Ludvig J.; Neuendorffer S.; Sachs S.; and Yuhong Xiong, 2003. *Taming heterogeneity - the Ptolemy approach. Proceedings of the IEEE*, 91, no. 1, 127–144. doi:10.1109/JPROC.2002.805829.

Maler O.; Manna Z.; and Pnueli A., 1992. *From timed to hybrid systems.* In J.W. de Bakker; C. Huizing; W.P. de Roever; and G. Rozenberg (Eds.), *Real-Time: Theory in Practice*. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-47218-6, 447–484.

Nutaro J.J., 2020. *Toward a Theory of Superdense Time in Simulation Models. ACM Transactions on Modeling and Computer Simulation*, 30, no. 3. doi:10.1145/3379489.

Ofenloch A.; Schwarz J.S.; Tolk D.; Brandt T.; Eilers R.; Ramirez R.; Raub T.; and Lehnhoff S., 2022. *MOSAIK 3.0: Combining Time-Stepped and Discrete Event Simulation.* In *2022 Open Source Modelling and Simulation of Energy Systems (OSMSES)*. 1–5. doi:10.1109/OSMSES54027.2022.9769116.

Zeigler B.P.; Muzy A.; and Kofman E., 2019. *Theory of Modeling and Simulation.* Elsevier. doi:10.1016/C2016-0-03987-6.