Modeling Time for the Execution of Heterogeneous Models

Frédéric Boulanger  Cécile Hardebolle  Christophe Jacquet
Iuliana Prodan

Abstract

Heterogeneous models mix different models of computation to model the behavior of a system. These models of computation may use different notions of time and different time scales. This article presents a modeling framework for representing relationships between the occurrences of events in the execution of heterogeneous models. Events may be triggered by other events or by elapsed time. Each event is modeled by a clock, and occurrences of the event are modeled by ticks. A clock has a time scale, which is the domain of the tags associated with its ticks to model the time at which an event occurs. We also present an algorithm for solving relations between clocks and determining which ticks should be considered as simultaneous and be handled as “now” in the next simulation step.
## History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-09-03</td>
<td>1.0</td>
<td>Initial version</td>
<td>Frédéric Boulanger</td>
</tr>
</tbody>
</table>

## Authors

<table>
<thead>
<tr>
<th>Author</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frédéric Boulanger</td>
<td>Supélec E3S</td>
</tr>
<tr>
<td>Cécile Hardebolle</td>
<td>Supélec E3S</td>
</tr>
<tr>
<td>Christophe Jacquet</td>
<td>Supélec E3S</td>
</tr>
<tr>
<td>Iuliana Prodan</td>
<td>Universitatea Politehnica din Bucuresti</td>
</tr>
</tbody>
</table>
## Contents

1. Introduction .................................................. 1  
2. Related Works ................................................. 1  
   2.1 MARTE ................................................... 1  
   2.2 CCSL ................................................... 1  
   2.3 The Tagged Signal Model (TSM) ........................... 2  
3. Reasons for a New Model of Time .............................. 2  
4. The Tagged Events Specification Language ................. 2  
   4.1 Clocks and Ticks ......................................... 3  
   4.2 Implication Relations .................................... 3  
   4.3 Tag relations ........................................... 7  
5. Solving TESL specifications .................................... 8  
   5.1 Implication and Tag Relations .......................... 8  
   5.2 Time Islands ........................................... 10  
   5.3 Greedy Clocks .......................................... 11  
   5.4 The Global Solving Algorithm ......................... 12  
   5.5 Error Detection ........................................ 12  
6. Conclusion ................................................... 12
1 Introduction

The growing power of modeling tools allows the design and verification of complex systems. This complexity leads to the joint use of several modeling paradigms in a given model, because different parts of the system belong to different technical domains, different abstraction levels require different modeling techniques, different aspects of the system belong to different domains, and different phases in the design of the system require different tools.

It is therefore necessary to be able to cope with heterogeneous models, and to give them a semantics which is precise enough to be able to execute them in order to simulate the behavior of the system, to analyze their properties in order to build valid systems, and to generate as much as possible of the realization of the system from the model. In previous work, we presented an approach of the execution of heterogeneous component-based models which relies on models of computation to describe the semantics of modeling paradigms, and on semantic adaptation to model the interactions between heterogeneous parts of a model. In order to execute models, this approach uses models of execution, which compute one possible execution of a model among all the executions that are allowed by its model of computation. Semantic adaptation between heterogeneous parts of a model is considered along three aspects: the adaptation of data, the adaptation of control, and the adaptation of time.

The adaptation of data consists in transforming data from a modeling paradigm to data in another paradigm at the boundary between heterogeneous parts of a model. For instance, an event with a value may be transformed in an input symbol for a state machine according to some decoding table.

The adaptation of control describes when different parts of the model should be given control, so that they can process their inputs and produce outputs. This can be quite difficult because the requirements of different models of computation for the execution of a model can interfere in complex ways. For instance, a periodically sampled signal processing task should be given control only at its sampling instants, not when some input becomes available for it. Some components may require that all of their inputs be available before they are given control. On other components, the relationship between the availability of inputs and control may be conditional (for instance, a logical OR gate needs to know the value of its second input only when the other input is false). Control may also depend on a mix of the occurrence of an event and meeting a deadline, for instance when a state machine has both regular and timed transitions leaving its current state.

Last, the adaptation of time takes care of modeling the different time scales used in different parts of an heterogeneous model. It can be used to model the relationship between the crankshaft and camshaft rotation angles in an engine, or to model the different paces at which time flows in GPS satellites and in a GPS receiver on ground as predicted by general relativity.

2 Related Works

2.1 MARTE

MARTE is a UML profile for the Modeling and Analysis of Real-Time Embedded systems. It contains a sub-profile which is dedicated to the modeling of time. The time sub-profile of MARTE is able to model the defects of real clocks compared to ideal clocks (for instance jitter). It relies on both synchronous and asynchronous relations between instants on different clocks (coincidence, precedence). It also supports polymorphic time: a clock may measure degrees of rotation, meters, persons entering a room, or seconds.

2.2 CCSL

CCSL is a language for specifying constraints on clocks. It relies on a model of time similar to the model of MARTE, but clocks have no time scale. The only way to measure time in CCSL is by counting ticks. It is not possible to specify a place in time on a clock if there is no tick at this time on the clock. CCSL comes with a model-checker named \texttt{Time}^2 (pronounced \textit{Time square}), which detects inconsistencies in a clock model, or builds a possible time-line if the specification is consistent. In previous work, we used CCSL to model the semantic adaptation of control between a discrete event model, a state machine and a synchronous dataflow model. However, without the possibility to specify the occurrence of an event at a given time, we could not model the occurrence of an event with an arbitrary delay after the occurrence of a triggering event.

The possibility to use precedence constraints between clocks in CCSL (and not only coincidence constraints) makes it very suitable for modeling models of computation. A model of computation only states the properties of the execution of a model, it does not force one particular execution. Therefore, CCSL can help validating a model independently of the execution platform on which it will run.
2.3 The Tagged Signal Model (TSM)

The Tagged Signal Model is a framework for modeling heterogeneous notions of time. It models a signal as a series of samples, and attaches a time tag to each sample. The tags for each signal belong to a time domain, and the properties of the time domain (is it a metric space, is it discrete, continuous, dense) give the properties of time. Morphisms between time domains can be used to model the semantic adaptation of time.

3 Reasons for a New Model of Time

The model of time in MARTE is well founded and complete, however, it is quite complex and is not supported by verification of simulation tools.

CCSL provides a formally defined subset of the model of time of MARTE, and provides a model-checker to verify clock specifications. Its declarative nature is also an advantage for the incremental refinement of the definition of models of computation. CCSL is therefore very close to our needs but two points are not compatible with our goal:

- There is no support for time scales. Ticks can only be positioned before, at, or after other ticks, and there is no notion of duration other than counting ticks on a clock.
- The relations between clocks are given once and for all, and at the time when we started this work, it was not possible to use the solver for computing a step by step execution of a model, and to force the occurrence of events provided by the inputs of the model. We need a solver that reacts at runtime to the ticks that occur because of the behavior of the components of the model. We want to use clocks only to model the semantic adaptation of time and control. The behavior of the components of the system can be modeled using other formalisms that are more suitable for describing computations on data for instance.

We therefore need a mix of a restriction of CCSL and a Tagged Signal Model. In the next section, we present our model of time, which was built to allow us to:

- specify coinciding occurrences of events (like CCSL)
- specify relations between time scales (like MARTE and the TSM)
- create occurrences of events
- solve a set of event occurrences according to the specification by determining which occurrences belong to the current instant, and what is the time tag for each occurrence of an event.

4 The Tagged Events Specification Language

Since our goal is to run discrete simulations of heterogeneous models, we need to specify what will happen at the next simulation step of the model. We call this next simulation step “now”. Each event is modeled by a clock, which has a tag domain. An occurrence of the event is modeled by the presence of a tick on the clock. A tick may have a tag, which means that we know when it should occur. If a tick has no tag, it will occur as soon as possible according to the relations between clocks. Such a tick is called a floating tick. A tick can also be set to “now”, which means that it belongs to the current instant, which corresponds to the next execution step of a model.

Ticks that are “now” may imply the creation of ticks on other clocks through coincidence relations. For instance, when clock $a$ implies clock $b$, which models the fact that an occurrence of the event modeled by $a$ always implies an occurrence of the event modeled by $b$, the existence of a “now” tick on $a$, implies the existence a “now” tick on $b$.

When ticks have tags, the relations between the tags of their clocks may imply coincidences. For instance, if there is a tick with tag 2 on clock $a$ and a tick with tag 10 on clock $b$, and the tags on $b$ are four times the tags on $a$ plus two, then we know that these two ticks coincide. Therefore, if any of these ticks is set to “now”, the other one should be set to “now” too.

In the following, we present the key concepts of TESL, the tagged events specification language.
4. THE TAGGED EVENTS SPECIFICATION LANGUAGE

4.1 Clocks and Ticks

In TESL, a Clock $c$ has a time domain $\text{dom}(c)$, with a total order $<$ on it. We note $C$ the set of clocks, and $D = \bigcup_{c \in C} \text{dom}(c)$ the union of the time domains.

A time context, which corresponds to the notion of instant, associates a clock with a totally ordered set of ticks. The order on the ticks is induced by the order on the time domain of the clock. No two ticks may bear the same tag. The special tag $\bot$ is used to denote “as soon as possible”, and is considered lower than any other tag. Among the ticks of a clock, at most one tick may be “now”.

A time context can therefore be modeled as a triplet $k = \langle \lambda, \tau, \nu \rangle$, with:

- $\lambda$: $C \rightarrow \mathbb{N}$
  
  $\lambda(c)$ is the number of ticks on clock $c$.

- $\tau$: $C \times \mathbb{N} \rightarrow D \cup \{\bot\}$
  
  $\tau(c, i)$ is the tag of the $i$th tick on clock $c$. $\tau$ has the following properties:
  
  1. $\forall c \in C, \forall i \in \mathbb{N}, \tau(c, i) \in \text{dom}(c) \cup \{\bot\}$
     
     The tags of the tick of a clock belong to the time domain of the clock.
  
  2. $\forall c \in C, \forall i > \lambda(c), \tau(c, i) = \bot$
     
     $\tau$ associates a tag only with existing ticks on $c$.
  
  3. $\forall c \in C, \forall (i, j) \in \mathbb{N}^2, i < j \Rightarrow \tau(c, i) < \tau(c, j)$
     
     Ticks on a clock are in increasing order, $\bot$ being considered as less than any other value in $D$.

- $\nu$: $C \rightarrow \mathbb{N}$

  $\nu(c)$ is the index of the “now” tick of $c$, or 0 if $c$ does not have a “now” tick.

  $\forall c \in C, \nu(c) \leq \lambda(c)$, the “now” tick is an existing tick of $C$.

Figure 1: Example clock

Figure 1 shows a clock with three ticks. The first one has no tag (it is tagged with $\bot$), and the other two have tags 10 and 16. The second tick is set to “now” and shown in black.

The interpretation of such a clock is that:

- the event modeled by this clock should occur as soon as possible;
- the event modeled by this clock should occur at time 10 and at time 16;
- the event that occurs at time 10 is in the current instant (now)

When using this model of time for driving the execution of a model, we need to know the tag of each tick of each clock. In this example, merging the two first ticks of this clock removes the floating tick and matches the original specification: “as soon as possible” is “now”, and the time tag of this instant is 10.

4.2 Implication Relations

CCSL has coincidence and precedence relations between clocks. For instance a clock $b$ may be a subclock of another clock $a$, which means that anytime there is a tick on $b$, there is a tick on $a$. Such a relation is acausal, which means that if there is a tick on $b$, there must be a tick on $a$ at the same time, and if there is no tick on $a$ there cannot be any tick on $b$ at the same time.

In TESL, we keep only coincidence relations because we want to be able to compute one possible execution of a model, and we use causal implication relations because we have to propagate the consequence of known occurrences of events at runtime instead of determining a set of possible solutions to the relations between clocks.
Therefore, instead of specifying that $b$ is a subclock of $a$, we choose to specify that $b$ implies $a$. Such a relation is directed, which fits our needs, and makes the specification much easier to solve since we only have to propagate the presence of ticks through implication relations instead of solving a system of equations.

Implication relations have a set of master clocks, which are on the left side of the implication, and one slave clock, on which “now” ticks are implied. An implication relation has a state, which summarizes the history of its master clocks. We can describe an implication relation as a sextuplet $I = \langle M, s, \Sigma, \varphi, \sigma_0, \delta \rangle$, where $M$ is the set of master clocks of the relation, $s$ is its slave clock, $\Sigma$ is the finite set of states or the relation, $\varphi : \{M\} \times \Sigma \to \mathbb{B}$ is the implication function of the relation, which tells whether a “now” tick should be put on the slave clock, $\sigma_0 \in \Sigma$ is the initial state of the relation, and $\delta : \{M\} \times \Sigma \to \Sigma$ is the transition function of the relation.

An implication relation can be considered as a process which creates a “now” tick on its slave clock when the master clocks have matched a pattern of event occurrences. Any implication relation described with a state machine can be used in TESL. For now, we provide the four following implication relations as examples:

- **$m$ implies $s$**

  In any time context $\langle \lambda, \tau, \nu \rangle$, if $m$ has a “now” tick, then $s$ has a “now” tick in the same time context. $m$ is the only master clock, $s$ is the slave clock. This relation is therefore described as: $\langle \{m\}, s, 1, \varphi, \star, \text{Id}_1 \rangle$, where $1$ is the singleton set with a unique element $\star$, and $\varphi = (\nu(m) \neq 0)$.

  ![Figure 2: Example of $m$ implies $s$](image)

- **$m$ filtered by $p$ implies $s$**

  In any time context $\langle \lambda, \tau, \nu \rangle$, if a “now” tick on $m$ makes pattern $p$ match, then $s$ has a “now” tick. The pattern $p$ is restricted here to the form: \([\text{skip, keep, rep-skip, rep-keep}]\), were skip is the initial number of “now” ticks to skip, keep is the number of consecutive “now” ticks to keep after the skipping phase, and rep-skip and rep-keep give a repeating pattern of ticks to skip and to keep. For instance, the pattern $[3, 2, 4, 1]$ will skip the 3 first “now” ticks, keep the next two, and then keep only one “now” tick out of five (skip four, keep one). $m$ is the only master clock, $s$ is the slave clock. This relation can be described as: $\langle \{m\}, s, [0, l - 1], \varphi, 0, \delta \rangle$, with:

  \[
  - l = \text{skip} + \text{keep} + \text{rep-skip} + \text{rep-keep}
  
  - \varphi(\{m\}, \sigma) = (\nu(m) \neq 0) \land
    \begin{cases}
    \text{skip} \leq \sigma < \text{skip} + \text{keep} \\
    \text{rep-skip} \leq \sigma - (\text{skip} + \text{keep})
    \end{cases}
  
  - \delta(\{m\}, \sigma) =
    \begin{cases}
    \text{if} \nu(m) = 0 \text{ then} \sigma \\
    \text{else if} \sigma < \text{skip} + \text{keep} + \text{rep-skip} \text{ then} \sigma + 1 \\
    \text{else} (\text{skip} + \text{keep}) + (\sigma + 1) \mod l
    \end{cases}
  \]

  ![Figure 3: Example of $m$ filtered by $[3,4,2,1]$ implies $s$](image)
4. THE TAGGED EVENTS SPECIFICATION LANGUAGE

Figure 3 shows two clocks $m$ and $s$ which satisfy the $m$ filtered by $[3,4,2,1]$ implies $s$ relation. The white ticks are ticks that may be “now” on $s$ for some other reason than this implication relation.

- $m$ delayed by $n$ on $c$ implies $s$

In any time context $(\lambda, \tau, \nu)$, if there is a “now” tick on $c$ which makes strictly more than $n$ “now” ticks on $c$ since the last “now” tick on $m$, then $s$ has a “now” tick. A “now” tick on $c$ which coincides with the “now” tick on $m$ is not taken into account. $m$ and $c$ are the master clocks ($c$ is the counting master clock), $s$ is the slave clock. This relation can be described as:

$$\langle \{m, c\}, s, [0, n], \varphi, 0, \delta \rangle$$

with:

- $\varphi(\{m, c\}, \sigma) = (\nu(c) \neq 0) \land (\sigma = 1)$
- $\delta(\{m, c\}, \sigma) = \begin{cases} n & \text{if } (\sigma > 0) \land (\nu(c) \neq 0) \text{ then } \sigma - 1 \\ \sigma & \text{endif} \end{cases}$

Figure 4: Exemple of $m$ delayed by 2 on $c$ implies $s$

Figure 4 shows three clocks $m$, $c$ and $s$ which satisfy the $m$ delayed by 2 on $c$ implies $s$ relation. The white ticks are ticks that may be “now” on $s$ for some other reason than this implication relation. We note $k_i$ the tick labelled $i$ on clock $k$. $s_1$ exists and is simultaneous with $c_2$ because two ticks were “now” on $c$ ($c_1$ and $c_2$) since the last “now” tick on $m$ ($m_1$). $c_4$ does not correspond to a “now” tick on $s$ because tick $c_3$ is simultaneous with $m_2$ and was not taken into account. We must wait until $c_5$ to have two ticks since $m_2$, and therefore a “now” tick on $s$ ($s_2$). $m_3$ starts a new counting sequence, but since $m_4$ occurs before the occurrence of two ticks on $c$, the count is reset, and we must wait until $c_8$ to get $c_3$.

- $m$ immediately delayed by $n$ on $c$ implies $s$

if there was $n$ “now” ticks on $c$ since the last “now” tick on $m$, then $s$ has a “now” tick. A “now” tick on $c$ which coincides with a “now” tick on $m$ is taken into account. This variant of the “delayed” relation can be described as: $\langle \{m, c\}, s, [0, n], \varphi, 0, \delta \rangle$, with:

- $\varphi(\{m, c\}, \sigma) = (\nu(c) \neq 0) \land \left( (\sigma = 1) \lor \left( (n = 1) \land (m.\nu \neq 0) \right) \right)$
- $\delta(\{m, c\}, \sigma) = \begin{cases} n - 1 & \text{if } (\nu(m) \neq 0) \land (\nu(c) \neq 0) \text{ then } n - 1 \\ \sigma & \text{endif} \end{cases}$

Figure 5: Exemple of $m$ immediately delayed by 2 on $c$ implies $s$

Figure 5 shows three clocks $m$, $c$ and $s$ which satisfy the $m$ immediately delayed by 2 on $c$ implies $s$ relation. The only difference with the previous example is that, since ticks on the counting
master clock are taken into account as soon as there is a “now” tick on \( m \), tick \( c_2 \) is simultaneous with \( c_4 \) in this case.

- **\( m \) sustained from \( b \) to \( e \) implies \( s \)**

  this implication relation is started by event \( b \) and stopped by event \( e \). A “now” tick on \( m \) implies a “now” tick on \( s \) only in time contexts that follow a time context where there was a “now” tick on \( b \) and there were non “now” tick on \( e \) since then. \( m \), \( b \) and \( e \) are the master clocks (\( b \) is the “begin” master clock, \( e \) is the “end” master clock), and \( s \) is the slave clock. This statement ignores “now” ticks on \( b \) and \( e \) that coincide with a “now” tick on \( m \), so the “filtering window” for sustaining \( s \) is \([b,e]\). Therefore, “now” ticks are put on the \( s \) clock for each “now” tick on the \( m \) clock which is strictly after a “now” tick on \( s \) and before of simultaneous to a “now” tick on \( e \). This relation can be described as: \( \langle \{m,b,e\},s,B,\varphi,\text{false},\delta \rangle \), with:

\[
- \varphi(\{m,b,e\},\sigma) = (\nu(m) \neq 0) \land \sigma
\]

\[
- \delta(\{m,b,e\},\sigma) = \begin{cases} 
\text{true} & \text{if } \nu(b) \neq 0 \\
\text{false} & \text{if } \nu(e) \neq 0 \\
\sigma & \text{else}
\end{cases}
\]

- **\( m \) sustained immediately from \( b \) to \( e \) implies \( s \)**

  this statement is similar to the previous one, but a “now” tick on \( b \) is taken into account even if it coincides with a “now” tick on \( m \), so the “filtering window” for sustaining \( s \) is \([b,e]\). The **immediately** keyword states that the implication starts immediately when there is a “now” tick on \( b \). This variant of the **sustained** relation can be described as: \( \langle \{m,b,e\},s,B,\varphi,\text{false},\delta \rangle \), with:

\[
- \varphi(\{m,b,e\},\sigma) = (\nu(m) \neq 0) \land (\sigma \lor (\nu(b) \neq 0))
\]

\[
- \delta(\{m,b,e\},\sigma) = \begin{cases} 
\text{true} & \text{if } \nu(b) \neq 0 \\
\text{false} & \text{if } \nu(e) \neq 0 \\
\sigma & \text{else}
\end{cases}
\]

- **\( m \) sustained from \( b \) to \( e \) weakly implies \( s \)**

  this statement is similar to the sustained statement, but a “now” tick on \( e \) is taken into account even if it coincides with a “now” tick on \( m \), so the “filtering window” for sustaining \( s \) is \([b,e]\). The **weakly** keyword states that the implication is weak and stops as soon as there is a “now” tick on \( e \). This variant of the **sustained** relation can be described as: \( \langle \{m,b,e\},s,B,\varphi,\text{false},\delta \rangle \), with:

\[
- \varphi(\{m,b,e\},\sigma) = (\nu(m) \neq 0) \land (\sigma \land (\nu(e) = 0))
\]

\[
- \delta(\{m,b,e\},\sigma) = \begin{cases} 
\text{true} & \text{if } \nu(b) \neq 0 \\
\text{false} & \text{if } \nu(e) \neq 0 \\
\sigma & \text{else}
\end{cases}
\]

- **\( m \) sustained immediately from \( b \) to \( e \) weakly implies \( s \)**

  this statement is similar to the sustained statement, but a “now” tick on \( b \) or \( e \) is taken into account even if it coincides with a “now” tick on \( m \), so the “filtering window” for sustaining \( s \) is \([b,e]\). This last variant of the **sustained** relation can be described as: \( \langle \{m,b,e\},s,B,\varphi,\text{false},\delta \rangle \), with:

\[
- \varphi(\{m,b,e\},\sigma) = (\nu(m) \neq 0) \land (\sigma \land (\nu(b) \neq 0) \land (\nu(e) = 0))
\]

\[
- \delta(\{m,b,e\},\sigma) = \begin{cases} 
\text{true} & \text{if } \nu(b) \neq 0 \\
\text{false} & \text{if } \nu(e) \neq 0 \\
\sigma & \text{else}
\end{cases}
\]
Since these implication relations are directed (contrary to CCSL clock relations), applying such implications to a set of clocks can only create "now" ticks. There is no mean in TESL to prevent the implication relations, the application of \( R \) in state \( \sigma_0 \) to \( k_0 \) yields a new time context \( k_1 \) which contains at least as many "now" ticks as \( k_0 \). Therefore, \( R_I \) in a given state \( \sigma \) can be considered as an increasing function on time contexts compared according to the number of "now" ticks. Since there can be at most one "now" tick on a given clock, the maximum number of "now" ticks in a time context is \( \text{card}(C) \), so there exists an integer \( f \) such that \( k_f = R_I(\sigma_f)^f(k_0) = k_{f-1} \). \( k_f \) is the fixed point of the implication relations in state \( \sigma_0 \), and \( f \leq \text{card}(C) \) since at least one "now" tick is created each time \( R_I \) is applied before reaching the fixed point.

What we call the application of the implication relations in state \( \sigma = (\sigma_1, \ldots, \sigma_m) \) to time context \( k_i = (\lambda_i, \tau_i, \nu_i) \) is \( R_I(\sigma)(k_i) = k_{i+1} \), with \( k_{i+1} = (\lambda_{i+1}, \tau_{i+1}, \nu_{i+1}) \) such that, for every clock \( c \in C \):

- \( \lambda_{i+1} = \begin{cases} \lambda_i(c) & \text{if } (\nu_i(c) \neq 0) \lor (\forall r_j = (M, c, \Sigma, \varphi, \sigma_0, \delta) \in R_I, \varphi(M, \sigma_{r_j}) = \text{false}) \\
\lambda_i(c) + 1 & \text{if } (\nu_i(c) = 0) \land (\exists r_j = (M, c, \Sigma, \varphi, \sigma_0, \delta) \in R_I, \varphi(M, \sigma_{r_j}) = \text{true}) \end{cases} \)

- \( \nu_{i+1} = \begin{cases} 0 & \text{if } (\nu_i(c) = 0) \land (\forall r_j = (M, c, \Sigma, \varphi, \sigma_0, \delta) \in R_I, \varphi(M, \sigma_{r_j}) = \text{false}) \\
1 & \text{if } (\nu_i(c) \neq 0) \lor (\exists r_j = (M, c, \Sigma, \varphi, \sigma_0, \delta) \in R_I, \varphi(M, \sigma_{r_j}) = \text{true}) \end{cases} \)

- \( \tau_{i+1}(c, 1) = \lambda_{i+1}(c) \lor (\forall j \in \mathbb{N}, \tau_{i+1}(c, j) = \tau_i(c, j)) \)

The first point states that no tick is added by implication relations to a clock which already has a "now" tick, or for which no relation implies a "now" tick, and that a tick is added to a clock which has no "now" tick and for which at least one relation implies a "now" tick.

The second point states that implication relations do not set a tick to "now" on a clock if no relation implies it, and that they do not unset an existing "now" tick.

The third point states that implication relations do not change the tag of existing ticks, but set the tag of an implied "now" tick to \( \perp \).

It is important to note that when computing the fixed point of \( R_I \), we do not update the state \( \sigma \) of the relations. Indeed, we are computing a time context which is the next instant of the clocks, so time does not advance during this computation, and the state of the implication relations should therefore not change. Once the time context which corresponds to the state of the clocks at the next instant

Implication relations allow us to model the control of \textit{event triggered} behaviors. For instance, in a hierarchical model, control must be given to the container of a component each time control is given to this component. This is necessary so that the container can provide the component with up-to-date inputs and process its outputs. Using TESL to model this, we would say that the control clock of the component implies the control clock of its container.

### 4.3 Tag relations

In order to model the control of \textit{timed-triggered} behaviors, we can create ticks with a tag which is the triggering time of the event. However, different clocks may use different sets of tags, and the tags may
advance at different rates on different clocks. TESL has tag relations to represent fixed conversions between time scales.

For two given clocks \( a \) and \( b \), a tag relation is a pair \( T = (d, r) \) of two functions \( d : \text{dom}(a) \to \text{dom}(b) \) and \( r : \text{dom}(b) \to \text{dom}(a) \). \( d \) is the direct conversion function and \( r \) is the reverse conversion function between tags on clocks \( a \) and \( b \).

In order to preserve causality, \( d \) and \( r \) must be increasing functions. The meaning of such a relation is that for any time context \( \sigma = (\lambda, \tau, \nu) \) any ticks \( t_a \) on \( a \) and \( t_b \) on \( b \), if \( d(\tau(a, t_a)) = \tau(b, t_b) \) or \( r(\tau(b, t_b)) = \tau(a, t_a) \), \( t_a \) and \( t_b \) are considered simultaneous. Therefore, if \( \nu(a) = t_a \), we should also have \( \nu(b) = t_b \) and reciprocally. However, \( d \) and \( r \) are not necessarily the inverse of each other. For instance, if \( \text{dom}(a) = \mathbb{R} \times \mathbb{N} \), which corresponds to super-dense time, with micro-steps numbered with integers to model a fine-grained causality between events that occur at the same real-time date, and \( \text{dom}(b) = \mathbb{R} \), we could have \( d : (t, n) \mapsto t \) and \( r : t \mapsto (t, 0) \), which are both increasing but are not the inverse function of each other.

The current implementation of TESL supports only affine tag relations, where \( d(t) = \alpha t + \beta \) with \( \alpha > 0 \), and \( r(t) = \frac{1}{\alpha} t - \frac{\beta}{\alpha} \). which are uniquely defined by constants \( \alpha \) and \( \beta \).

Contrary to implication relations, tag relations cannot create new ticks on clocks. However, they can set existing ticks to “now” if their tag matches the tag of a “now” tick on another clock through the transitive closure of the tag relations.

4.3.1 Consistency of tag relations

Tag relations can be represented as an undirected graph, with clocks as vertices. This undirected graph can be considered as the superposition of two directed graphs: one with the direct conversion functions as edges, the other with the reverse conversion functions as edges. When the current tag is unknown for a clock \( b \), it can be computed if the current tag is known on a clock \( a \) and there is a path between \( a \) and \( b \) in the undirected graph. This path may be made of edges from the direct conversion graph and of edges from the reverse conversion graph. So, converting the current tag on \( a \) into the current tag on \( b \) is made using the \( d \) and \( r \) functions of different tag relations.

If there is several paths from \( a \) to \( b \) in the graph, they should all give the same tag for \( b \). When this is property holds for any pair of clocks, the tag relations are consistent. Consistency only requires that if, starting from a known tag on a clock \( a \), we compute the corresponding yet unknown tag on a clock \( b \), we find the same result along all possible paths. However, if the tag on \( b \) is already known, the computed value may be different of the known one, simply because \( d \) and \( r \) in tag relations are not the inverse function of each other. However, each tag relation must hold either according to \( d \) or according to \( r \).

Let us consider again the previous example of \( a \) and \( b \) with \( \text{dom}(a) = \mathbb{R} \times \mathbb{N} \) and \( \text{dom}(b) = \mathbb{R} \), and a tag relation \( r = (d : (t, n) \mapsto t, r : t \mapsto (t, 0)) \). If we know that the current tag on \( b \) is \( t_0 \), we can compute the corresponding tag on \( a \) as \( r(t_0) = (t_0, 0) \). However, if the current tag on \( a \) is also already known to be \( (t_0, k) \), there is no problem because the tag relation holds according to \( d \).

When the set of tag relations is consistent, for each clock \( a \), we note \( d_a : \text{dom}(a) \times C \to D \cup \{\bot\} \) the function which maps a tag on \( a \) to the corresponding tag on any other clock \( b \) if there is a path from \( a \) to \( b \) in the tag relation graph, or \( \bot \) if there is none. We note \( r_a : C \times D \to \text{dom}(a) \) the function which maps a tag on any clock \( b \) to the corresponding tag on \( a \) if there is a path from \( b \) to \( a \) in the tag relation graph, or \( \bot \) if there is none.

5 Solving TESL specifications

A TESL specification consists of a set of clocks, a set of implication relations, a set of tag relations and an initial time context \( k_0 \) and implication relation state \( \sigma_0 \). This context gives ticks to clocks, tags to ticks, and may set at most one tick to “now” on each clock.

Starting from this initial context, we want to determine for each clock, what is the tag of each of its ticks and if one of this ticks is “now”.

5.1 Implication and Tag Relations

The first step is to use the implication relations in state \( \sigma_0 \) in order to create the “now” ticks that are implied by the existing ones. As seen in section 4.2 this is done by applying the implication relations until a fixed point is reached and no new tick is created.

Then, we must find the tag of each tick. This is done by using the \( d \) and \( r \) functions which correspond to the transitive closure of the tag relations. For each clock \( a \) that has a “now” tick with
a tag $t_a$, and for each clock $b$ such that either $\hat{d}_{ab}(t_a, b) \neq \bot$ or $\hat{r}_{ba}(a, t_a) \neq \bot$ (there exists a relation between the time scales of clocks $a$ and $b$), we compute the tag $t_{ba}$ which corresponds to $t_a$ on clock $b$. We have either $t_{ba} = \hat{d}_{ab}(t_a, b)$ or $t_{ba} = \hat{r}_{ba}(a, t_a)$ If $b$ has a tick with tag $t_{ba}$, this tick is set to “now” because it is simultaneous with the “now” tick on $a$. If $b$ has a floating (without tag) “now” tick, its tag is set to $t_{ba}$ for the same reason.

Setting a tick to “now” may have consequences through the implication relations. Therefore, after computing tags and determining “now” ticks using the tag relations, the implication relations must be applied again until no new tick is set to “now” in the process. If we note $R_I$ the application of the implication relations to a set of clocks, and $R_T$ the application of the tag relations, the set of clock obtained thus far is the fixed point of $R_T \circ R_I$ reached when starting from the initial set of clocks $C_0$.

### 5.1.1 Example of solving implication and tag relations

![Figure 7: Example TESL specification](image)

Figure 7 shows an initial TESL specification and the three steps that lead to the fixed point of this specification. Please note that contrary to the chronograms used to illustrate implication relations, these diagrams show the state of clocks at a given step in the solving algorithm. The curved arrows on the right show the relations between the three clocks. Solid arrows represent implication relations, dashed arrows represent tag relation. The double arrow on tag relations shows the orientation which corresponds to the direct conversion function shown aside.

The first diagram shows the initial time context: clock $a$ has a “now” tick with tag 1, clock $b$ has no tick, and clock $c$ has a tick with tag 3. The implication relation from $a$ to $b$ implies that there is a “now” tick on $b$, so in step 1, $b$ has a floating “now” tick (its tag is not known yet). No other implication relation can be applied, so we will now use tag relations.

The affine tag relation between clocks $a$ and $b$ says that the tags $t_a$ and $t_b$ of two simultaneous ticks on $a$ and $b$ verify $t_b = 3t_a - 1$. Since the floating tick on $b$ is “now”, it is simultaneous with the
tick on \( a \) which has tag 1. Therefore, this tick gets tag 2, as shown on step 2.

The affine tag relation between clocks \( a \) and \( c \) says that the tags \( t_a \) and \( t_c \) of two simultaneous ticks on \( a \) and \( c \) verify \( t_c = 2t_a + 1 \). Since the tick on \( c \) has tag \( 3 = 2 \times 1 + 1 \), it is simultaneous with the tick on \( a \) which has tag 1. Therefore, this tick becomes “now”, as shown in the final clocks. No other relations can be applied and all ticks have a tag, so the algorithm stops. The next simulation step will run with an occurrence of \( a \), an occurrence of \( b \) and an occurrence of \( c \), at times 1, 2 and 3 on their respective time scales.

\[
\begin{align*}
\text{Initial clocks} & \quad \text{After implications} & \quad \text{After merging} \\
\begin{array}{c}
\text{\( a \)}
\end{array} & \begin{array}{c}
\text{\( b \)}
\end{array} & \begin{array}{c}
\text{\( c \)}
\end{array} \\
\hline
1 & \text{\( \times 2 + 1 \)} & \text{\( \times 2 + 1 \)} \\
3 & \text{\( \times 3 - 2 \)} & \text{\( \times 3 - 2 \)} \\
5 & \text{\( \times 3 - 2 \)} & \text{\( \times 3 - 2 \)} \\
7 & \text{\( \times 3 - 2 \)} & \text{\( \times 3 - 2 \)} \\
\end{array}
\end{align*}
\]

5.2 Time Islands

It may happen that, in the fixed point reached by applying the implication and tag relations to the initial set of clocks, some clocks have a floating tick that is “now” and also a tagged tick. The floating “now” tick may have been created because of an implication relation, and the tagged tick is merely a part of the specification that requests an event occurrence at a given time. Since something must happen “now” (because of the floating “now” tick) and it was requested that something happen at a given time (the tagged tick), it may seem natural to merge the floating and the tagged tick. However, doing so may change the current time on other clocks through tag relations.

Figure 8: Example with two time islands and a merge
The consequences of the merging of a floating and a tagged tick are limited to the connected component of the graph of the tag relations to which the clock belongs. Clocks that belong to different connected components have unrelated time scales. There may be implication relations between such clocks, but there are no relation between the tags of their ticks. We call such connected components time islands.

The problem of merging floating and tagged ticks can be solved independently in each time island of the set of clocks. However, in each time island, if several merges are possible, it is important to perform only the one that yields the smallest tag on each clock of the time island. By doing so, we ensure that time advances not more than requested by the specification, and that we don’t move existing tagged ticks to “the past” by choosing a too large tag for “now”.

The merge step therefore consists in looking, in each time island, for the merge operation that gives the smallest tag to the merged ticks. This is done by examining in turn each clock $c_i$ that has a potential merge, and by computing the tag that would be “now” on each other clock $c_j$ of the island if the merge was performed (using either $d_{c_i}$ or $r_{c_j}$). Since tag relations are increasing functions, we can compare any clock to compare the tags that correspond to the different possible merges, and choose the smallest one. We then perform the corresponding merge.

Figure 8 shows four clocks $a$, $b$, $c$, and $d$ which belong to two time islands $\{a, b\}$ and $\{c, d\}$. Initially, only clock $a$ as a “now” tick with tag 1. It also has a tick with tag 3. Clock $b$ has a tick with tag 5, clock $c$ has a tick with tag 7, and clock $d$ has a tick with tag 5. Clock $a$ implies clock $b$, which implies clock $c$, which in turn implies clock $d$. There are tag relations between $a$ and $b$, and between $c$ and $d$. The tags of clocks $c$ and $d$ are not related to the tags of clocks $a$ and $b$, so we have two time islands.

Applying the implication relations creates floating “now” ticks on $b$, $c$, and $d$. Using the tag relation between $a$ and $b$, we can determine that the floating “now” tick on $b$ should have tag 3, that the tick with tag 3 on $a$ corresponds to tag 7 on $b$, and that the tick with tag 5 on $b$ corresponds to tag 2 on $a$. Therefore, setting the tag of the floating tick on $b$ to 3 gives a correct solution because the “now” ticks have the smallest tags.

Using the tag relation between $c$ and $d$ shows that the tick with tag 7 on $c$ corresponds to tag 3 on $d$, and that the tick with tag 5 on $d$ corresponds to tag 13 on $c$. The smallest tags is the pair $7c$, $3d$, so we can merge the floating “now” tick on $c$ with the tick tagged 7, and set the tag of the floating “now” tick on $d$ to 3. It would have been incorrect to merge the floating “now” tick on $d$ with the tick tagged 5 because this would have put the tick tagged 7 on $c$ into the past.

5.3 Greedy Clocks

When a clock $c$ has a tagged tick, but the transitive closure of the tag relations does not allow to compare this tag to the tag of any “now” tick in the set of clocks, we can choose to make this tick “now” or not. Such a situation happens when there is no “now” tick in the time island to which $c$ belongs. Making such a tick “now” amounts to making the simulation advance as fast as possible according to the specification. Choosing not to set it to “now” amounts to making the simulation advance no faster than required by the specification.

We can also consider that setting such ticks to “now” leads to the greatest fixed point of the relations, while leaving them for a future simulation step leads to the least fixed point of the relations.

However, this choice is not possible for all clocks, because stating that an existing tick is “now” may not match the actual semantics of the event which is modeled by the clock. For instance, a real-time periodic clock may have a tick which is tagged with the date of the next time this clock should tick. However, this tick should become “now” only when this date is reached in real time. It should be set to “now” only by the process that handles real time, not by the solver of the clock specification. Therefore, only clocks that are explicitly stated as greedy may have their ticks forcibly aligned with the current instant. Such clocks are called “greedy” because their try to maximize the number of “now” ticks.

5.3.1 A Use of Greedy Clocks: Driving Clocks

When running a simulation, the execution of the model of the system must be driven by some causes. It can be an external source of input events for the model, a periodic sampling of the environment, or even a loop that runs the simulation as fast as possible.

In order to synchronize the execution of a model with these triggering events, we model them as driving clocks. A driving clock always has a tick, which corresponds to the next occurrence of the driving event. This tick becomes “now” when the corresponding event really occurs, for instance when the user clicks on a button, when the period of the driving clock has elapsed, or as soon as possible.
An issue is to know on which driving clock a simulation depends. The execution engine loads a
model and starts running the simulation, but how can it determine for which events it should wait
before computing the next simulation step?

Greedy driving clocks provide a handy solution to this issue. By solving the TESL specification of
the model with greedy driving clocks, the solver will find that these clocks have a tick (they always
have one, which corresponds to the next occurrence of their event), but that no “now” tick can be
used through tag relations to make them “now” (this is because driving clocks are the primary cause
of things happening in the model). If the driving clocks are greedy, the solver will set their tick to
“now” if the specification allows it. By examining the solved set of clocks, we can determine all the
driving clocks that are a potential trigger for the next simulation step because they have a “now” tick.

Once these driving clocks are known, the simulation engine just waits for one of these events to
occur, and solve the specification again, but this time with non-greedy driving clocks. Indeed, we are
now looking for the consequences of events that really occurred, so the solver is not allowed to set a
tick to “now” if the corresponding event didn’t occur. On the contrary, when looking for the potential
causes of the next simulation step, making the driving clocks greedy allows the solver to find the
sources of all causality chains that lead to a valid set of “now” ticks with respect to the specification.

5.4 The Global Solving Algorithm

We have seen and justified the different steps for determining which ticks are “now” and to compute
their time tag. We will now assemble all the pieces together.

Given a set \( C \) of clocks, a set \( R_I = \{r_1 \ldots r_m\} \) of implication relations, and a set \( R_T = \{t_1 \ldots t_n\} \) of tag relations, we start with an initial time context \( k_0 \), and a state \( \sigma_0 \) for the relations. Applying \( R_I(\sigma_0) \) to \( k_0 \) yields a new time context \( k_1 \) and a state \( \sigma_1 \) for the implication relations. For each relation \( r_i = (M, S, \Sigma, \varphi, \sigma_0, \delta) \), \( \sigma_1_i = \delta(M, \sigma_0_i) \), and a floating tick is added to \( S \) in \( k_1 \) if \( \varphi(M, \sigma_0_i) \) is true. Applying all the \( R_i \) in state \( \sigma_0 \) until no new tick is added leads to the fixed point \( k_z \) of the implication relations.

Then, the transitive closure of the tag relations is used to compute the tag of ticks that are
simultaneous with “now” ticks whose tag is already known, and to set to “now” ticks that have a tag
which corresponds to the tag of a “now” tick on another clock.

The next step consists, in each time island, in merging a floating “now” tick with a tagged tick,
therefore setting the current time on the island. The choice of the merged tick is done in order to
minimize the date of “now”, to avoid putting future ticks prematurely in the past.

Then, if a greedy clock as a tagged tick which is not “now” and whose tag cannot be related to the
tag of any other “now” tick, this tick is set to “now”.

Since merging ticks and aligning the time of greedy clocks to now sets ticks to “now”, we start the
whole algorithm again until no new tick is set to “now”.

We stop when no ticks is set to “now” by the algorithm, and get the final time context \( k_f \). The
state \( \sigma_1 \) computed by the implication relations in context \( k_f \) will be used as the initial state of the
implication relations when computing the fixed point time context for the next simulation step.

5.5 Error Detection

Inconsistencies in the TESL specification are detected during solving. Different kinds of errors can be
found:

- if a “now” tick has no tag at the end of the algorithm, the specification is not complete enough.
- if a tick has a lower tag that a “now” tick on the same clock, the specification is inconsistent.
- if two tags are not in the same order than the corresponding tags on another clock, the tag
relations are not causal.

However, these errors are detected by the solving algorithm at runtime. We don’t have a static error
detection algorithm, and we don’t known if such an algorithm exists.

6 Conclusion

TBD
Bibliography


