Towards multi-layered temporal models:
A proposal to integrate instant refinement in CCSL

Mathieu Montin[1,2] [0000−0003−2219−9359] and Marc Pantel[2] [0000−0001−7591−0402]

[1] INRIA, Team Veridis, University of Lorraine, LORIA, Team MOSEL
[2] IRIT, ENSEEIHT, INPT, Team ACADIE

Abstract. For the past 50 years, temporal constraints have been a key driver in the development of critical systems, as ensuring their safety requires their behaviour to meet stringent temporal requirements. A well established and promising approach to express and verify such temporal constraints is to rely on formal modelling languages. One such language is CCSL, first introduced as part of the MARTE UML profile, which allows the developer, through entities called clocks, to abstract any system into events on which constraints can be expressed, and then assessed using TIMESQUARE, a tool which implements its operational semantics. By nature, CCSL handles horizontal separation (component based design at one step in the system development) of concerns through the notion of clocks, but does not yet take into account the other major separation of concerns used in modern system development: vertical separation, also called refinement in the literature (relations between the various steps of the system development). This paper proposes an approach to extend CCSL with a notion of refinement in order to handle temporal models relying on both vertical and horizontal parts. Our proposal relies on the notion of multi-layered time to provide two new CCSL relations expressing two different yet complementary notions of refinement. Their integration with the other CCSL constructs is discussed and their use is illustrated while the relevance and future impacts of this extended version of CCSL is detailed.

Keywords: CCSL· refinement · temporal constraints.

1 Introduction

1.1 Ubiquity of complex systems

Software engineering as a discipline has been developed since the middle of the 20th century. The overall consensus is that the 1960s, and especially the 1968 convention held in Marktoberdorf, Germany [2] paved the way to where software engineering stands now: present in every single field and company, as well as in our everyday life. While we believe that the future induced by this evolution will be bright, it can only be so provided we have the ability to validate and verify the software being made, in other words, to assess that it satisfies its user requirements.

This need is especially potent when considering critical systems, the failure of which could have significant negative impacts on human lives. These critical systems usually rely on various interactions with other entities: other software through logical interfaces, human beings through user interfaces and the outside world through both sensors, which

provide the software with data, and actuators, which control various physical mechanisms. The software from the last category are called cyber physical systems (CPS) [6]. The actuators in these systems have their output constantly adjusted using the data received from their sensors, and such adjustments must usually be done in real time, thus inducing various hard temporal constraints. These critical systems are especially present in fields such as aeronautics, cars, railway, space exploration, energy transformation, healthcare and factory automation, which means they have the potential to be of huge size and complexity, and their modelling usually needs to be heterogeneous.

1.2 Heterogeneous modelling

Complexity of description is a major issue in the development and especially in the verification of such software. This complexity, as described in the theory of complex systems [15] is wide and encompasses a large number of possible cases. It was first called algorithmic complexity [7] by its inventors Kolmogorov and Chaitin and was originally related to the size of the simplest algorithm in a given language that can implement the system. This notion is nowadays wider, as the system in question can either be huge in size, and provide a sophisticated service, composed of numerous small size elements which are required to interact with one another in a correct manner, or simply display a very complex behaviour which translates into a deep conceptual challenge for the developers. Such complexity is very concrete, especially in CPS where it can often be found in its various incarnations.

While developing and assessing such complex systems, the notion of separation of concerns becomes mandatory, as first introduced by Dijsktra himself in [5]. A concern is a specific element of a system that must be handled during its development. As for separation of concerns, while the name being somewhat self-explanatory, it hides a higher level of complexity depending on the nature of the concerns that are separated from others. Indeed, these concerns can be of various nature: functional, physical, logical, abstraction, human, sociological, ergonomic, psychological, economical, ethical... and can refer to various macroscopic elements: provided service, provided quality of service... including the process, methods and tools for the development itself.

There are three main kinds of separation of concerns that are usually called horizontal, transversal and vertical. Horizontal separation of concerns (also called Component Based Design) consists in applying a divide and conquer strategy that splits a problem in sub-problems recursively until reaching problems that are small enough to be solved efficiently. The complexity of the remaining components is reduced as their size is small, however they usually mix different issues to be solved: security, functionality, performance... The second kind of separation, called transversal, corresponds to Aspect Oriented Engineering. It proposes to isolate each of the previous issues to handle them separately with the most appropriate tools. The third kind is called vertical separation: it consists in separating a given development into different steps from an abstract specification to a concrete implementation through a process usually called refinement.

1.3 Handling vertical and horizontal separation over temporal constraints

Our work revolves around the handling of the behavioural aspect of a system development, that is, the specific aspect-oriented subpart of a system concerning temporal constraints over its behaviour. These temporal constraints are not absolute in our work, which means there will be no mention of worst-case execution time (WCET) in this work, but rather they are relative from one component to another. The word "component" is not used lightly: this indeed corresponds to the various components horizontal separation has revealed and defined, and the way these components interact time-wise with one another. This horizontal description is usually expressed using languages such as CCSL, which will be described in more details in Section 2.3. This language relies on a notion of time and time structure that will be summarized in Section 2.1. However, this kind of description do not handle vertical separation, which means temporal constraints between layers of refinement cannot be expressed. In this paper, we propose a way to express such constraints, and we motivate the need for such expressiveness. A first step in that direction consists in using a multi-layered time structure that will be described in more details in Section 2.2. Then, new CCSL relations can be defined between several layers of refinement, thus allowing relations to be cross-specifications. We call the first new relation 1-N refinement, which is described in Section 5, while the second relation, called 1-1 refinement, is depicted in Section 6. Both these relations will be formally defined, illustrated by examples, and integrated to CCSL through properties of preservation with existing CCSL relations. An example of system which could benefit from such a multi-layered temporal description is depicted in Section 3.

2 Theoretical ground

This section presents the state-of-the-art elements that we rely on in this work. They consist of common notions about time, as well as one of our previous works regarding multi-layered temporal structures and basic notions about CCSL.

2.1 Time, partial orders and time structures

In order to model the temporal relations between instants in the execution of complex systems, the usual approach is to use strict partial orders which allow some instants to be coincident and some others to be unrelated, as opposed to total orders where all instants must be related one way or another. A strict partial order is a mathematical structure composed of four entities, which are the following:

- A set of instants, I.
- A first relation over the elements of I, $\_\approx\_$, called coincidence.
- A second relation over the elements of I, $\_<\_$, called precedence.
- A proof that $\_\approx\_$ and $\_<\_$ satisfy the properties of strict partial ordering.

The last element of this structure ties all the others together. By giving the right properties to the relations, it also gives them the intended semantics, that is: $\_<\_$ is a strict precedence between elements of I and $\_\approx\_$ is a relation of equivalence between its elements. These properties are as follows:

1. _≈_ is an equivalence relation
   - _≈_ is reflexive — $\forall i \in I : i \approx i$
   - _≈_ is transitive — $\forall (i,j,k) \in I^3 : i \approx j \wedge j \approx k \Rightarrow j \approx k$
   - _≈_ is symmetrical — $\forall (i,j) \in I^2 : i \approx j \Rightarrow j \approx i$
2. _<_ is irreflexive towards _≈_ — $\forall (i,j) \in I^2 : i < j \Rightarrow \neg\, i \approx j$
3. _<_ is transitive — $\forall (i,j,k) \in I^3 : i < j \wedge j < k \Rightarrow j < k$
4. _<_ respects _≈_
   - on the left — $\forall (i,j,k) \in I^3 : i \approx j \wedge i < k \Rightarrow j < k$
   - on the right — $\forall (i,j,k) \in I^3 : i \approx j \wedge k < i \Rightarrow k < j$

As an example of modelling with such structure, let us consider Alice and her usual morning routine: Alice gets up, after which she either takes a bath first followed by eating or vice versa. She always sings while in the bath. After that, she takes off for work. The two possible traces depicting her behaviour over a single day are shown in Figure 1. They consist of the following possible events: getting up (u), bathing (b), singing (s), eating (e) and taking off (o).

(a) A first possible behaviour (b) A second possible behaviour

Fig. 1: Both possible behaviours

These possible behaviours can be merged using a time structure[16], with an underlying partial order, that is depicted on Figure 2. The events b and e are concurrent and are not linked by any of the two relations composing the strict partial order. The blue vertical dashed line represents coincidence (when events occur simultaneously) while the red arrows represent precedence (one occurs strictly before the other). Note that, while we arbitrary chose to represent e before b, it does not mean that e precedes b, and e could equally have been placed somewhere else between u and o.

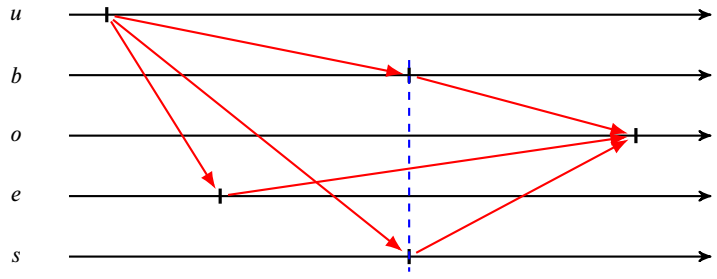


Fig. 2: Alice's morning routine time structure

2.2 Multi-layered time structures

Such time structures naturally embed a notion of horizontal separation. Indeed, each of the events it contains is, by definition, separated from the others, regardless of the origin of said events. In other words, whether these events come from the same system or from different systems that are being coordinated is not relevant. However, all these events are from the same step in the development, in other words, they are from the same layer of refinement. This level of observation is fundamental when dealing with such systems because a trace or a time structure only takes into account observable events. For instance, in Alice's morning routine, one did not chose to represent details about her different activities, such as the way she showers (from hair to toes) or the sequence of songs she sings while doing so. These more precise events could have been depicted in another time structure. In a previous work [10], we showed that, should we assign a partial order to any of these depictions (any of these levels of refinement), then these partial orders ($(<_c, \approx_c)$ and $(<_a, \approx_a)$) would be related with one another in a certain formal manner, as follows:

Let Ω be the set of all sets: $\forall I \in \Omega, \forall (<_c, <_a, \approx_c, \approx_a) \in (I \times I)^4$:
$(<_c, \approx_c) <_r (<_a, \approx_a) \overset{d}{\iff}$

$$\forall (i,j) \in I : \begin{pmatrix} i <_c j \Rightarrow i <_a j \vee i \approx_a j & (1) \\ i <_a j \Rightarrow i <_c j & (2) \\ i \approx_c j \Rightarrow i \approx_a j & (3) \\ i \approx_a j \Rightarrow i \approx_c j \vee i <_c j \vee j <_c i & (4) \end{pmatrix}$$

In this definition, the level annotated by the index c is the lowest (the more concrete) level of observation and a is the highest (the more abstract). We state what it means for a pair of relations to refine another pair of relations. We can only compare pairs of relations that are bound to the same underlying set of instants. This relation is composed of four predicates, each of which indicates how one of the four relations is translated into the other level of observation.

- *Precedence abstraction:* If a strictly precedes b in the lower level, then it can either be coincident to it in the higher level or still precede it. This means that a distinction which is visible at a lower level can either disappear at a higher one or remain visible, depending on the behaviour of the refinement for these instants – *Equation (1)*
- *Precedence embodiment:* If a strictly precedes b in the higher level, then it can only precede it in the lower level. This means that the distinction between these instants already existed in the higher level and thus cannot be lost when refining. Looking closer at a system preserves precedence between instants – *Equation (2)*
- *Coincidence abstraction:* If a is coincident to b in the lower level, they stay coincident in the higher level. Looking at the system from a higher point of view cannot reveal temporal distinction between events – *Equation (3)*
- *Coincidence embodiment:* If a is coincident to b in the higher level then these instants cannot be independent in the lower level ; they will still be related but nothing can be said on the nature of this relation – *Equation (4)*

2.3 Horizontal constraints with CCSL

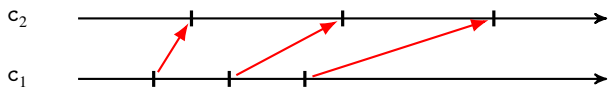
A time structure, such as the one depicted in Figure 1 displays relations between the occurrences of events. A time structure can either be seen as a specification, depicting how occurrences of events should be related, or as a temporal implementation of an informal specification depicting how the instants are bound to one another to respect a certain set of relations. Said relations can be expressed informally, but it is usually preferred to use a formal modelling language dedicated to this purpose. CCSL [1] is such a language, that was defined in the MARTE [13] UML [12] profile. It proposes various relations and expressions to bind events (that are named clocks in the language) with one another. Considering two clocks c_1 and c_1, here are the CCSL relations that will be used in the rest of the paper to illustrate our contribution:

1. *Subclocking*: $c_1 \sqsubseteq c_2 \iff \forall i \in c_1, \exists j \in c_2, i \approx j$

2. *Equality*: $c_1 \backsim c_2 \iff c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_1$

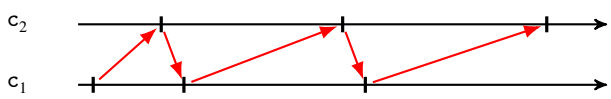
3. *Exclusion*: $c_1 \sharp c_2 \iff \forall (i,j) \in (c_1 \times c_2), \neg i \approx j$

4. *Precedence*: $c_1 \prec c_2 \iff \exists f \in (c_2 \rightarrow c_1), \forall i, (f\,i) < i$



5. *Causality*: $c_1 \preccurlyeq c_2 \iff \exists f \in (c_2 \rightarrow c_1), \forall i, (f\,i) \leq i$

6. *Alternation*: $c_1 \lesssim c_2 \iff c_1 \prec c_2 \wedge \forall i, j \in c_2^2, i < j \Rightarrow i < (f\,j)$



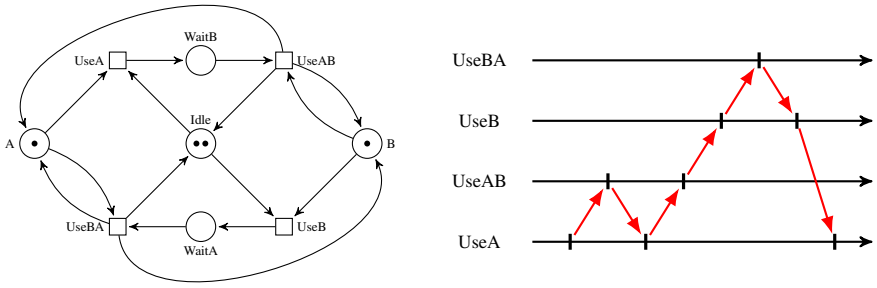
These definitions were first introduced in [3] and were mechanized in a previous work [9]. Note that the precedence and causality definitions have been simplified for the purpose of this paper, as there are more constraints to the binding function f (it has to be bijective for instance). These additional constraints are naturally taken into account in the formal mechanized counterpart of this work. All details can be found in [8].

3 An example of multi-layered modelling

This theoretical ground provides us with the following expressiveness: it is possible to specify the behaviour of several sub-parts of a given system, as well as several parts from different systems. It is also possible to compare orders with one another towards the notion of refinement, but it is not yet possible to express constraints and properties between events coming from different layers of refinement. This section proposes a simple example as to why this is relevant, after which solutions will be proposed.

3.1 The *Deadlock* Petrinet

Petri nets [14] are considered by many as the assembler of concurrent system, and their semantics is well known in the field. Let us consider the Petrinet on Figure 3a, usually used as a toy example for teaching purposes.



(a) Representation of the net

(b) A possible execution of the net

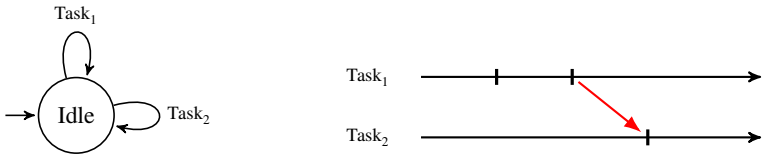
Fig. 3: The *Deadlock* Petrinet

This net, however simple, is very interesting because it allows a state of deadlock to be reached. It consists of two processes, initially in the *Idle* state, that can execute two tasks: the first one requires the use of the resource A, then the resource B after which, both resources are freed, and the second task is similar, although the resources are required in the opposite order. Both these tasks can be done by any of the two processes, a possibly infinite number of time, as long as both resources are freed when the second process begins the other task. If both processes require the use of their first resource concurrently, a deadlock state is reached because both processes will endlessly wait for the resource that is currently retained by the other process. A possible execution of this

net resulting in a deadlock is depicted in Figure 3b. After two cycles of the first task and one cycle of the second one, resource *B* is required by a process while resource *A* is required by the second one, hence the deadlock.

3.2 A functional view of the system

As surprising as it may seem, writing constraints on this system to forbid the deadlock case is not trivial. In CCSL for instance, it is not easy to forbid intertwining of events. In this specific example, although it is possible, expressing that an occurrence of *UseB* should never occur between an occurrence of *UseA* and *UseAB* – and vice versa – requires a long list of constraints.



(a) The abstract automaton (b) A possible execution of the automaton

Fig. 4: A functional representation of the *Deadlock* net

An easier way of expressing such constraints arises when considering the abstract function the deadlock-free runs of this systems fulfil. In this case, such a function is simple: the net should execute both tasks an arbitrary number of times, in an arbitrary order, regardless of the resources actually used in these tasks. Such a behaviour can be depicted as an automaton, shown on Figure 4a, with a possible execution of the system shown on Figure 4b.

3.3 Binding the two levels of description

The interesting aspect of this second system is that it does not allow any faulty behaviour, which the first one does. Should we synchronize these systems, we would be able to forbid such deadlocked runs. The usual manner to coordinate systems with one another is to compose them horizontally, as seen in Section 2.3. However, this implicitly means that both systems are part of a global system whose behaviour has to be specified. In this case, this is not true because they do not live in the same level of observation. The second one can be seen as a specification while the first one can be seen as a possible implementation of said specification. In other words, expressing the fact that the *Deadlock* net should behave as an instantiation – a more concrete implementation – of the automaton should ensure the correctness of its behaviour. This means that constraints between layers of refinement need to be expressed in such cases. Both levels should have their own layer of time, and both these layers should satisfy the relation depicted in Section 2.2. In each of these layers, constraints specific to each system can be written and expressed using CCSL. The remaining step is to allow the definition of inter-layers

constraints, binding each abstract clock with concrete clocks that refine the event it represents. In our example, this would mean expressing that both *UseA* and *UseAB* should refine $Task_1$ while *UseB* and *UseBA* should refine $Task_2$, which the following sections will make possible.

4 Stakes of the approach

On the combination of CCSL *and refinement* In Section 2.2 we summarized a previous work on the modelling of refinement which proposes to handle the different layers of refinement by assigning each of them a separate partial order and ordering these orders with a specific relation. CCSL is itself based on partial orders which means both our notion of refinement and CCSL share the same formalism, which allows us to mix them together to assess how CCSL would behave when combined with refinement.

On preservation over instants Before starting the investigation on this conjunction, it is important to note that the notion of refinement depicted in Section 2.2 is fundamentally between partial orders. In that regard, it preserves by nature any required property over these, because it has been defined in that purpose. For instance, the strict precedence between instants is preserved through embodiment using the second equation, while the coincidence between instants is preserved through abstraction as depicted by the third equation. In other words, there is no need to prove that our refinement relation preserves the right properties in terms of instants ordering, because it does so by definition.

On preservation over clocks However, these preservations are natural in terms of relations between instants, yet not for relations between clocks. This means that it would be a mistake to assume that, by nature, relations between clocks should be preserved by the use of our relation of refinement. This makes the following question relevant: are there some properties between clocks which, when specified at a given level of refinement, could be transferred into another level of refinement without additionnal requirements ? Sections 5 and 6 aim at answering this question after introducing refinement relations between clocks.

An example of what to expect Let us take an example of what to expect here: at a given level of refinement, we know that subclocking is transitive. This means that, given three clocks c_1, c_2 and c_3, if we know that $c_1 \sqsubseteq c_2$ and $c_2 \sqsubseteq c_3$ then we can deduce $c_1 \sqsubseteq c_3$. In other words, the property $c_1 \sqsubseteq c_3$ does not need to be given as an additional constraint, because it is deducible from the rest of the context. In the following sections, we try to assess such assumptions, but in various levels of refinement.

Refinement between clocks In order to establish such properties, we need to express what it means for clocks to refine one another. In that purpose, we propose two different relations of refinement between clocks, both of which bind clocks from different levels of refinement. The first one, depicted in Section 5 considers a refinement with an arbitrary number of refined ticks for a given abstract event, whereas Section 6 depicts a more constrained form of refinement, where a single concrete tick is allowed. These two notions are motivated both by their own expressiveness and by the various CCSL relations they preserve.

Temporal context We consider, for the remaining of this paper, two layers of refinement characterized by two partial orders $(<_c, \approx_c)$ and $(<_a, \approx_a)$. We assume that these partial orders satisfy $(<_c, \approx_c) <_r (<_a, \approx_a)$. In that context, whenever a CCSL relation will be mentioned, it will be prefixed with the layer of refinement in which it is defined. For instance, if a clock c_1 precedes a clock c_2 in the abstract layer of refinement, it will be written $c_1 \prec_a c_2$. Usually, the level of abstraction of each clock will also be written, in which case the previous relation becomes $c_{a_1} \prec_a c_{a_2}$. In this context, preservation properties toward other existing CCSL constructs can and will be discussed. As a side node, every result that is given afterwards has been mechanized and proved using the AGDA proof assistant [11], even though said mechanization will not be detailed in this paper. The full development is provided in the first author PhD report [8].

5 A first generic CCSL relation of refinement

5.1 Definition of 1-N refinement

Intuition 1-N refinement aims at modelling the most common, unconstrained relation of refinement. In that purpose, each abstract clock can be refined by several concrete clocks, while each concrete event must be abstracted by a single abstract clock. In addition, each tick of the abstract clock can be refined by a strictly positive number of ticks for each of its concrete clocks. These number of ticks can vary throughout the execution of the system. The following example and definition will emphasize and capture these informal requirements.

Example Let us consider the following situation: a worker is driving nails in a plank of wood. In the abstract level, we consider driving the nail as an atomic action, while in the more concrete level we consider hitting the nail with the hammer the atomic action. Depending on how strong the worker is, it might take him a few strikes for each nail to be driven in the plank, and each nail could be driven in a different number of strikes. In this case, the clock $Striking_c$ is a 1-N refinement of the clock $Driving_a$, as shown on Figure 5, where the blue rectangles are the equivalence classes from the abstract point of view. Note that, although this picture is very similar to the ones in Section 2.3, both clocks are here coming from different levels of abstraction.

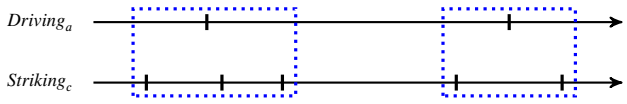


Fig. 5: An example of 1-N refinement

Definition As a formalization of the previous example and intuition, we define a relation of 1-N refinement between clocks, with C being the set of all clocks:

$$\forall (c_c, c_a) \in C^2, \\ c_c \lesssim_{1-n} c_a \Leftrightarrow (\forall i \in c_a, \exists j \in c_c, i \approx_a j) \wedge (\forall j \in c_c, \exists i \in c_a, i \approx_a j)$$

This definition is composed of two parts as follows:

- Any tick of the abstract clock is refined by a tick of the concrete clock. These two ticks are coincident from the abstract point of view.
- Any tick of the concrete clock is a refinement of a tick of the abstract clock. These two ticks are coincident in terms of the abstract partial order.

The required unicity of the abstraction in terms of ticks is a direct consequence of this definition. Indeed, should we take two abstract ticks of the same concrete ticks, they are coincident from the abstract point of view by transitivity of the abstract coincidence, since both are coincident with the concrete tick. Moreover, clocks are subsets of totally ordered instants which means that two coincident instants of the same clock are in fact propositionally equal, which ensures the required unicity. Such a property, along with others, is part of the formal counterpart of this work.

5.2 1-N refinement and coincidence-based CCSL relations

We experiment how 1-N refinement behaves when combined with CCSL notions related to coincidence, that is subclocking, equality, exclusion and union.

Subclocking As coincidence between instants is preserved through abstraction, one would expect subclocking to be preserved as well. This has been proven to hold, which lead to the following theorem:

$$\forall (c_{c_1}, c_{c_2}, c_{a_1}, c_{a_2}) \in C^4, \\ (c_{c_1} \lesssim_{1-n} c_{a_1}) \wedge (c_{c_2} \lesssim_{1-n} c_{a_2}) \wedge (c_{c_1} \sqsubseteq_c c_{c_2}) \Rightarrow (c_{a_1} \sqsubseteq_a c_{a_2})$$

Equality Since equality between clocks is a case of double subclocking, equality is also preserved through abstraction, which leads to the following theorem:

$$\forall (c_{c_1}, c_{c_2}, c_{a_1}, c_{a_2}) \in C^4, \\ (c_{c_1} \lesssim_{1-n} c_{a_1}) \wedge (c_{c_2} \lesssim_{1-n} c_{a_2}) \wedge (c_{c_1} \backsim_c c_{c_2}) \Rightarrow (c_{a_1} \backsim_a c_{a_2})$$

Exclusion Refining excluded clocks cannot create coincident instants, which means the refined clocks are excluded as well. This makes sense because the abstract excluded clocks have ticks that are all in different equivalence classes regarding abstract coincidence and the refined instants are still in these classes and thus cannot be coincident even from the lower point of view. This leads to the following theorem:

$$\forall (c_{c_1}, c_{c_2}, c_{a_1}, c_{a_2}) \in C^4, \\ (c_{c_1} \lesssim_{1-n} c_{a_1}) \wedge (c_{c_2} \lesssim_{1-n} c_{a_2}) \wedge (c_{a_1} \sharp_a c_{a_2}) \Rightarrow (c_{c_1} \sharp_c c_{c_2})$$

Multiple concrete clocks When two clocks refine the same one, it means that these two clocks track events that are part of the events tracked by the clock being refined. Thus, it is natural to assume that the union of these clocks is still a refinement of the abstract clock. In CCSL, the union of two clocks is simply the union of their ticks, which leads to the following theorem:

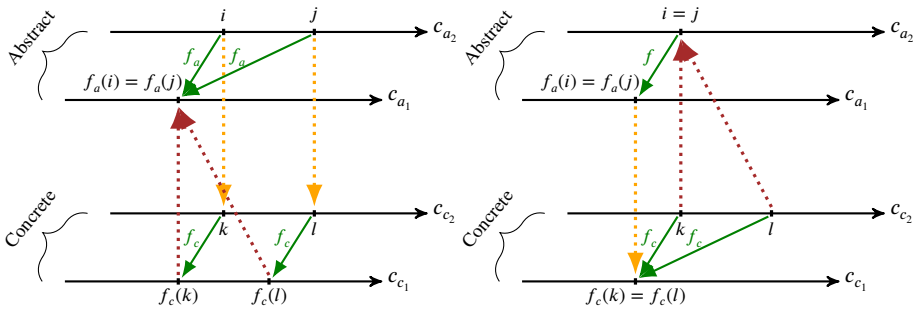
$$\forall (c_c, c_{c_1}, c_{c_2}, c_a) \in C^4, \\ (c_{c_1} \lesssim_{1-n} c_a) \wedge (c_{c_2} \lesssim_{1-n} c_a) \wedge (c_c = c_{c_1} \cup c_{c_2}) \Rightarrow (c_c \lesssim_{1-n} c_a)$$

Multiple abstract clocks On the other hand, when a clock is a refinement of two clocks, this means that the event it tracks is deduced from two entities, leading us to assume that these entities are in fact the same. And indeed, our formalism implies such clock equality, showing that our clock refinement is not symmetrical, as expected. Here is the related theorem:

$$\forall (c_c, c_{a_1}, c_{a_2}) \in C^3, \\ (c_c \lesssim_{1-n} c_{a_1}) \wedge (c_c \lesssim_{1-n} c_{a_2}) \Rightarrow (c_{a_1} \backsim_a c_{a_2})$$

5.3 1-N refinement and precedence-based CCSL relations

While 1-N refinement preserves CCSL notions related to coincidence, as depicted in Section 5.2, investigating its impact on notions based on precedence (precedence, causality and alternation) is not as fruitful since proving the refinement is not sufficient for these relations to be transported. Figure 6 shows two examples where these relations are not preserved. On these pictures, the two levels of abstraction are represented, as well as two clocks per level. The brown arrows represent the abstraction of an event while the orange ones show an embodiment of such event, following the relation of refinement between these clocks. The functions f_a and f_c are the binding functions of the precedences from both levels. Note that the instants i, j, k, l are not suffixed because they do not belong to any specific level of abstraction, or rather, they belong to both.



(a) Clock precedence and abstraction (b) Clock precedence and embodiment

Fig. 6: 1-N refinement and precedence

Abstraction of precedence Figure 6a shows an example where a concrete precedence ($c_{c_1} \prec_c c_{c_2}$) is not preserved through abstraction ($\neg(c_{a_1} \prec_a c_{a_2})$). This happens because two instants that precede one another in the concrete level (in this case $f_c(k)$ and $f_c(l)$) might be two instances of the same abstract event. In this case two events that precede one another in this abstract level (i and j) might actually be mapped to the same abstract event by the precedence function f_a we are trying to build, which should be injective.

Embodiment of precedence Figure 6b shows an example where an abstract precedence ($c_{a_1} \prec_a c_{a_2}$) is not preserved through embodiment ($\neg(c_{c_1} \prec_c c_{c_2})$). This happens for a similar reason that invalidates the injectivity of the function f_c.

Causality and alternation Since alternation is a specific case of precedence and causality is a less constrained case of precedence, both of these relations are necessarily not preserved through abstraction nor embodiment as well. The reason is that a tick can be refined by several ticks of the same clock, thus possibly compromising the injectivity of the binding function. Should we forbid such behaviour, more relations could be preserved, which leads to 1-1 refinement.

6 A second specific CCSL relation of refinement

6.1 Definition of 1-1 refinement

Intuition 1-N refinement does not provide an immediate preservation of coincidence-based CCSL relations, because it depicts a situation where such relations simply are not propagated from one level to the next. As stated before, the underlying reason of such a limitation does not lie in our ability to model refinement, but rather in the arbitrary number of ticks each concrete clock can have bound to a single abstract tick. By changing the number of ticks, relations around precedence are naturally not preserved because they rely on the bijectivity of the binding function. 1-1 refinement is meant to preserve the number of ticks through refinement. Each abstract clock can still be refined by an arbitrary number of concrete clocks, although each abstract tick can now only be refined by a single tick of each concrete clocks.

Example Going back at our example from Section 5.1, we can imagine that the worker can now drive nails more efficiently: he manages to do so in only one strike but it requires an increased accuracy, which is now considered a new concrete step for the purposes of this example. This new situation is depicted on Figure 7.

Definition This example leads to the definition of a relation of 1-1 refinement, which only differs from 1-N refinement by the uniqueness of the refined ticks:

$$\forall (c_c, c_a) \in C^2,\quad c_c \lesssim_{1-1} c_a \Leftrightarrow (\forall i \in c_a, \exists! j \in c_c, i \approx_a j) \wedge (\forall j \in c_c, \exists i \in c_a, i \approx_a j)$$

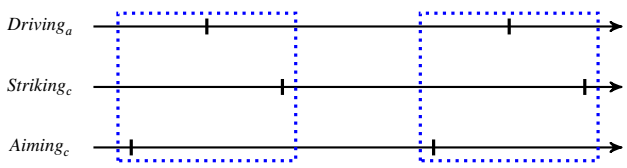


Fig. 7: An example of 1-1 refinement

6.2 1-1 refinement and coincidence-based CCSL relations

Since unique existence implies existence, 1-1 refinement is trivially a specific case of 1-N refinement, which means that any theorem regarding 1-N refinement and coincidence-based CCSL relation shown in Section 5.2 still holds for 1-1 refinement.

6.3 1-1 refinement and precedence-based CCSL relations

The main goal of the 1-1 refinement is to provide CCSL with a notion of clock refinement which naturally preserves precedence-related relations. In this section we investigate to what extent this preservation is guaranteed.

Embodiment of causality Causality is not preserved through embodiment, even with 1-1 refinement, and for a very concrete reason, which is that abstract coincidence can basically be transformed into any relation in the concrete level. Let us take four clocks c_{c_1}, c_{c_2}, c_{a_1} and c_{a_2} such that $c_{c_1} \lesssim_{1-1} c_{a_1}$ and $c_{c_2} \lesssim_{1-1} c_{a_2}$. If we have $c_{a_1} \preccurlyeq_a c_{a_2}$ this means that for a tick i of c_{a_2} we have a tick j of c_{a_1} such that $f(i) = j$ and $i \leq_a j$. It is thus possible that $i \approx_a j$ which, in the concrete level, can be transformed into $j \prec_c i$ which invalidates preservation of causality through embodiment.

Embodiment of precedence Precedence, however, does not exhibit this kind of possibility, and is preserved directly through embodiment, leading to the following theorem:

$$\forall (c_{c_1} c_{c_2} c_{a_1} c_{a_2}) \in C^4, \\ (c_{c_1} \lesssim_{1-1} c_{a_1}) \wedge (c_{c_2} \lesssim_{1-1} c_{a_2}) \wedge (c_{a_1} \prec_a c_{a_2}) \Rightarrow (c_{c_1} \prec_c c_{c_2})$$

Abstraction of precedence and causality Causality and precedence are both preserved in term of causality through abstraction. In other words, causality is fully preserved while precedence becomes causality, leading to the following theorem:

$$\forall (c_{c_1} c_{c_2} c_{a_1} c_{a_2}) \in C^4, \\ (c_{c_1} \lesssim_{1-1} c_{a_1}) \wedge (c_{c_2} \lesssim_{1-1} c_{a_2}) \wedge ((c_{c_1} \prec_c c_{c_2}) \vee (c_{c_1} \preccurlyeq_c c_{c_2})) \Rightarrow (c_{a_1} \preccurlyeq_a c_{a_2})$$

1-1 refinement and alternation Since precedence is transformed into causality through abstraction, we cannot expect alternation to be preserved in the process. As for embodiment, Figure 8 gives us an example as to why it does not hold either. Indeed, in the concrete level we can see that both $f_c(k)$ and $f_c(l)$ precede k, invalidating alternation.

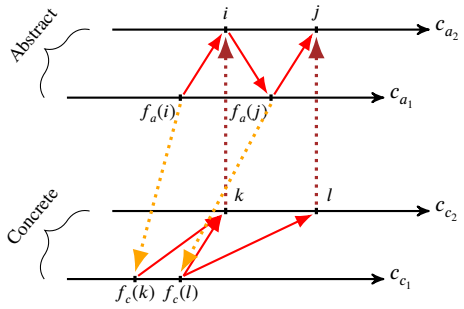


Fig. 8: Alternation and embodiment

7 Additional relations of refinement

An interesting question would be to assess whether other refinement relations would be relevant both in terms of expressiveness and in terms of direct preservation of properties, as even 1-1 refinement does not ensure the preservation of all CCSL constructs. This section briefly (and informally) explores this question.

1-X refinement 1-X refinement could mean both of the following ideas: either X is set and each abstract tick is refined by X ticks of the concrete clock, or X is not set in which case each abstract tick must be refined by the same number of concrete ticks. In the second case, the first ticks of the abstract clocks fixes X, after which the clocks behaves as in the first case, which makes both of them very similar. Both cases are a special case of 1-N refinement and could be modelled as such, although they would not improve the number of direct properties that are preserved, since they both change the overall number of ticks of the clocks. Thus 1-X refinement seems moderately relevant.

N-N refinement The notions of 1-1 and 1-N refinement revolves around the number of possible ticks rather than on the number of possible clocks. In both cases, these refinements are actually 1-N in terms of clocks. Although a N-N refinement in terms of clocks is easy to imagine (and is actually currently supported by considering the union of the N abstract clocks) it is hard to picture what a N-N refinement in terms of ticks would mean. This could mean that any number of ticks of the concrete clock would stand for any number of abstract ticks, which does not seem reasonable nor relevant when thinking in terms of concrete vs abstract. This could require further inspection but 1-1 and 1-N refinement as presented seem to capture the essence of behavioural refinement.

8 Conclusion

8.1 Assessments

We propose an extension of the CCSL modelling language in order to enable refinement checking between models. We provide two new CCSL relations expressing the

expected refinement between two clocks: 1-1 refinement and 1-N refinement. These relations are different from the others CCSL relations because they build a bridge between clocks coming from two different specifications rather than a bridge between clocks from the same specification. Each of these different specifications must correspond to a given level of refinement in the sense that their partial order should comply with the relation of refinement which was defined in Section 2.2. This extends the spectrum of CCSL through the expression of relations between models at the various development steps.

This extended CCSL can be used in multiple ways. It allows the developer to compare CCSL models in terms of refinement. It also provides a wide area of experimentation around refinement, since 1-1 refinement is more constrained but directly preserves most CCSL relation while 1-N refinement is more permissive but only preserves the main ones. Inside this range, our formal framework enables the definition and proof of preservation of other intermediate refinement relations less constraining than the 1-1 and preserving more properties than the 1-N. Moreover, this context can directly be used throughout the development of time critical systems using the classical top down design where a concrete model must refine an abstract one, in other words the concrete implementation must comply with the abstract specification. Finally, this work can be used to make explicit a common behaviour between various concrete models in the same way inheritance captures common aspects between different classes in object oriented design, ultimately enabling their synchronisation through this common behavioural interface.

In the process of adding refinement to CCSL, we investigated how CCSL operators behave when coupled with the new relations of clock refinement. These preservation properties have been mathematically proven in our formal framework that is thoroughly depicted in [8]. This investigation was fruitful when dealing with CCSL notions which are bound to coincidences using 1-N refinement, in the sense that refinement is fairly regular towards coincidence. It was even more fruitful when considering 1-1 refinement, which was designed to preserve properties of precedence, and which succeeded at doing so. These preservation properties are fundamental regarding the confidence we have in capturing the essence of refinement between clocks in a multi-layered temporal context.

8.2 Perspectives

TIMESQUARE [4] is an operational semantics for CCSL that provides us with a modelling environment with associated simulation and verification tools. It is based on a single partial order on which clocks are built conforming to a given specification. The main perspective for our work would be to enrich both the official version of CCSL and TIMESQUARE with our notions of refinement, so that actual engineers could design their multi-layered models in CCSL and verify them using TIMESQUARE. This requires to extend the core of the tool to handle multiple partial orders instead of one.

Our work could be used to specify relations between systems whose behaviours are similar, in the sense that they refine the same specification. We would like to emphasize and validate this aspect of our work through examples in that direction.

Our approach could be used on a wider range of case studies including more complex industrial size ones that would probably emphasize the relevance of our approach in modelling the behaviour of complex systems.

References

1. André, C., Mallet, F.: Clock Constraints in UML/MARTE CCSL. Research Report RR-6540, INRIA (2008)
2. Buxton, J.N., Randell, B.: Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. 1969, Brussels, Scientific Affairs Division, NATO (1970)
3. Deantoni, J., André, C., Gascon, R.: CCSL denotational semantics. Research Report RR-8628 (2014)
4. Deantoni, J., Mallet, F.: TimeSquare: Treat your Models with Logical Time. In: TOOLS - 50th International Conference on Objects, Models, Components, Patterns - 2012 (2012)
5. Dijkstra, E.W.: On the Role of Scientific Thought, pp. 60–66. Springer New York, New York, NY (1982). https://doi.org/10.1007/978-1-4612-5695-3_12, `https://doi.org/10.1007/978-1-4612-5695-3\_12`
6. Lee, E.A.: Cyber physical systems: Design challenges. In: 11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008), 5-7 May 2008, Orlando, Florida, USA. pp. 363–369. IEEE Computer Society (2008). https://doi.org/10.1109/ISORC.2008.25
7. Li, M., Vitányi, P.M.B.: An introduction to Kolmogorov complexity and its applications. Texts and Monographs in Computer Science, Springer (1993). https://doi.org/10.1007/978-1-4757-3860-5
8. Montin, M.: A formal framework for heterogeneous systems semantics. Ph.D. thesis (2020), `http://montin.perso.enseeiht.fr/these.pdf`
9. Montin, M., Pantel, M.: Mechanizing the denotational semantics of the clock constraint specification language. In: Abdelwahed, E.H., Bellatreche, L., Golfarelli, M., Méry, D., Ordonez, C. (eds.) Model and Data Engineering - 8th International Conference, MEDI 2018, Marrakesh, Morocco, October 24-26, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11163, pp. 385–400. Springer (2018). https://doi.org/10.1007/978-3-030-00856-7_26, `https://doi.org/10.1007/978-3-030-00856-7\_26`
10. Montin, M., Pantel, M.: Ordering strict partial orders to model behavioral refinement. In: Derrick, J., Dongol, B., Reeves, S. (eds.) Proceedings 18th Refinement Workshop, Refine@FM 2018, Oxford, UK, 18th July 2018. EPTCS, vol. 282, pp. 23–38 (2018). https://doi.org/10.4204/EPTCS.282.3
11. Norell, U.: Towards a practical programming language based on dependent type theory. Ph.D. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (Sep 2007)
12. (OMG), O.M.G.: Unified modeling language (Dec 2017), `https://www.omg.org/spec/UML/About-UML/`
13. (OMG), O.M.G.: UML profile for MARTE (Apr 2019), `https://www.omg.org/spec/MARTE/About-MARTE/`
14. Petri, C.A.: Kommunikation mit Automaten. Dissertation, Schriften des IIM 2, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Bonn (1962)
15. Thurner, S., Hanel, R., Klimek, P.: Introduction to the theory of complex systems. Oxford University Press (2018)
16. Winskel, G.: Event structures. In: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proc. of an Advanced Course, Bad Honnef, 8.-19. September 1986 (1986)