Guard Evaluation and Synchronization Issues in Causal Semantics for UML2.X Sequence Diagrams

Fatma Dhaou, Inès Mouakher, J. Christian Attiogbe, Khaled Bsaïes

Guard Evaluation and Synchronization issues in Causal Semantics for UML2.X Sequence Diagrams

Fatma Dhaou[1], Ines Mouakher [1], J.Christian Attiogbé[2] and Khaled Bsaies [1]
[1]*Lipah, Faculty of Sciences Tunis, Tunis, Tunisia*
[2]*LS2N, University of Nantes, Nantes, France*

Keywords: UML2.X Sequence Diagrams, Semantics, Combined Fragments, Guard Evaluation

Abstract: UML2.X sequence diagrams (SD) permit the modelling of behaviours of systems. With the combined fragments (CF) that can be nested and aggregated in a single SD more complex behaviours can be created. However the standard interpretation and the consideration of all the aspects related to the CF are not obvious to handle, this represses their proper use. We proposed in a previous work, a causal semantics based on partial order theory, which is suitable for the computation of all possible valid traces for UML2.X sequence diagrams with nested CF modelling behaviours of distributed systems. In this work, we deal with an important aspect that is the guard evaluation and the synchronization between the lifelines that are complex issues.

1 INTRODUCTION

Context. The context of our work is the use of the UML2.X sequence diagrams (SD) as a privileged language that are adopted by the engineering and designers in the modelling of behaviours of systems due its intuitive use and its intuitive graphical representation.
However, the standard semantics of SD that is defined by the Object Management Group (OMG) (Group, 2015) leaves several intentional variations points to adopt the use of this language in the modelling of different kinds of systems. These intentional variations points lead sometimes to ambiguities that induce some issues, which repress the proper use of SD.
Motivation. We interested especially on the computation of partial order between the events for UML2.X SD modelling behaviours of distributed systems. We reported the inadequacy and the ambiguities of the definitions of the standard semantics whose its rules did not take into account of the independence of the components involved into an interaction. For the other works, they proposed two approaches for the computation of the partial order. The first approach consists in flattening the SD into several basic SDs that are semantically equivalent ((Cengarle and Alexander, 2004), (Knapp and Wuttke, 2006), (øystein Haugen et al., 2005), (Harald, 2003)). The second approach uses the concept of locations that is borrowed from the Live Sequence charts formalism ((Cavarra and Filipe, 2004), (Juliana, 2006), (Harel and Maoz, 2008)).
We have considered the most popular CF SEQ, ALT, OPT and LOOP that permit to model respectively sequential, alternative, optional and iterative behaviours. The standard defines weak sequencing as the default composition operator for CF. Accordingly, most semantics retain this operator to compose a combined fragment with the rest of the diagram ((Harald, 2003), (øystein Haugen et al., 2005), (Cengarle et al., 2005)). However the weak sequencing operator can lead to non intuitive orders between events. This explains that the most of the existing semantics interpret the some of CF as in the structured programming language. For instance the ALT CF is considered as the *IF-Then-Else* construct and in the loop CF strict sequencing is applied between iterations. These non-standard interpretation limit considerably the expressive power of CF and some order between events is lost. Moreover, the most of the existing approach did not deal explicitly with nested CF whose complicate the determination of the precedence relations between the events. In our previous work, we proposed an extension of the causal semantics to remedy to the insufficiencies of the existing works and to deal with some issues of the standard semantics. Indeed we have firstly interested on computation of partial order between the events of UML2.X SD that model behaviours of distributed systems.

In this paper we interested on issues that are related to the consideration of constraints interactions

that guard some combined fragments. Indeed this aspect is often ignored by the most of the existing semantics, (Aredo, 2002), (Harald, 2003), (Li et al., 2004), (Grosu and Smolka, 2005), (Cengarle et al., 2005), since it causes several inconsistencies. The few semantics that have considered guards, have either proposed approaches that are not convenient for all kind of systems or they adopt some strict hypothesis. In the semantics where only synchronous messages are considered, the guard evaluation does not raise any issue. However, in the distributed systems where the communication is mainly asynchronous it must be handled with caution. Indeed, after the guard evaluation in the case of an ALT, a synchronization between the lifelines (that model the objects involved in the interaction) must be ensured in order to prevent a parallel execution of several operands.

Contribution. This paper extends our previous work (Dhaou et al., 2015), (Dhaou et al., 2016), (Dhaou et al., 2017). In (Dhaou et al., 2015), (Dhaou et al., 2016) we have extended an existing semantics proposed for UML 1.X (O.Tahir et al., 2005) to deal with SD with the most popular combined fragments (ALT, OPT and LOOP), by processing the SD as a whole (without parsing the SD). We have proposed several rules to derive the partial order between the events. In (Dhaou et al., 2017), we have updated the formalization of sequence diagrams in order to support nested CF, and we have generalized the previous rules for the derivation of partial order between events.

We now propose some additional contributions that consist, in proposing an approach to deal with guard evaluation and synchronization issue in our causal semantics that is dedicated to SD with nested CF modelling behaviours of distributed systems.

Organization. The remainder of the article is structured as follows. In Section 2, we propose an overview on our previous work. Section 3 is devoted to the approach that we propose for guard evaluation for SD with nested CF modelling behaviours of distributed systems. In Section 4, we define the behaviour of SD. Then in Section 5, we we deal with the synchronization issue. Before concluding in Section 7, we present some related works in Section 6.

2 OVERVIEW ON THE PREVIOUS WORK

The extended causal semantics (Dhaou et al., 2017) is based on precedence relations that permit to compute, for each event of an SD with nested CF its preceding events, without parsing the SD and without making syntactic restrictions on the considered CF (especially for LOOP CF or nested structure that contains LOOP CF that requires a special processing).

To define and formalize these precedence relations, we have proceeded as follows.

• We first proposed a formalization of UML2.X that is based on set theory notations. Thus an SD is defined as a tuple:
$SD : \langle L, M, EVT, FCT\_s, FCT\_r, FCT\_l, OP, F, <_{caus}, tree\_OP \rangle$
where the elements depict respectively the set of lifelines, the set of asynchronous messages, the set of events, the function that permits to associate to each message one sent event, the function that permits to associate to each message one received event, the function that associates to each event one lifeline (the transmitter or the receiver), the set of the operands, the set of the combined fragments, the causal relation and the function that allows to structure the SD in the form of a tree of operands,

• then we have encoded the SD into a tree structure, such that the tree is composed by a set of linked operands. Indeed, we consider the whole SD as a root operand that we note OP_{00}; we define the set $OP = (\bigcup_{i=\{1..n\}} OP_i) \cup \{OP_{00}\}$; where n is the number of operands of the considered SD. The figure 2 represents the tree associated to the SD of the figure 1. The idea consists to associate a node to each CF or operand. When building the tree of an SD, we always have a root node that represents the complete SD; the process is then breadth-first. Note that the operands of the CF ALT are independent, i.e they have disjoint executions. Therefore, to simplify the tree representation of the SD, we substitute the node which should stand for these fragments with the nodes representing their operands. They are moved to the upper level. However, to distinguish them, the operands of the same fragment have their indexes built with the same prefix (it's the case of the operands OP21 and OP22 that belong to the same CF ALT (see Fig.1). From the node of a current SD, the consecutive fragments of the SD become the nodes of the current node. Each fragment is either represented as a node or it is represented by the nodes of its operands. A node is associated to each CF that has only one operand (for instance loop or opt). A CF with more than one operand (for instance ALT CF) is replaced with the nodes associated to its operands.

Each operand in an SD has a weight. For instance, each operand of a SEQ, an ALT and an OPT CF have a weight equal to 1, and an operand of a LOOP CF has a weight equal to a value *max*, where *max* is the maximum number of iteration of the considered LOOP CF. The general definition of an operand in a

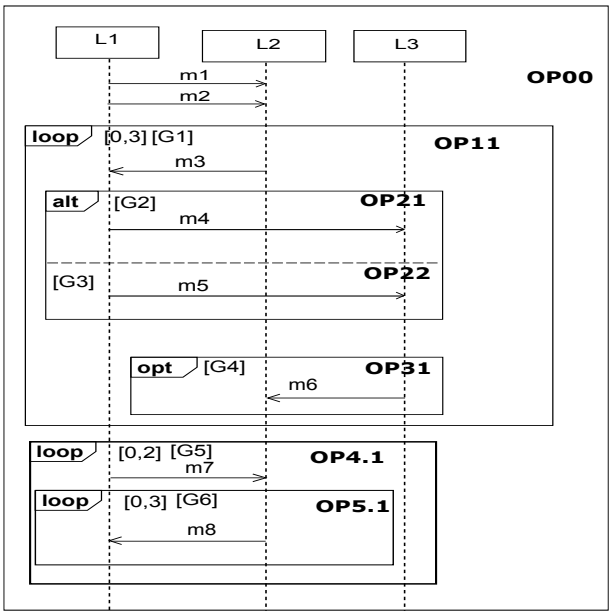

Figure 1: Example of SD with nested CF

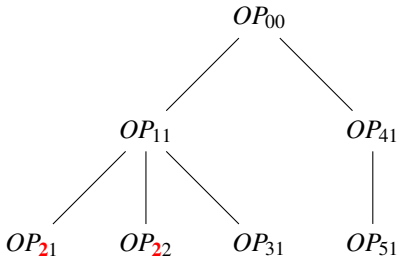

Figure 2: Tree associated to the SD of the figure 1

combined fragment is given as follows:

Definition 1. ***(Operand in Combined Fragment)*** *We define a set of operands OP_i in a CF F_i as follows:* $OP_i = \{OP_{i,j=\{1..k\}} \mid OP_{ij} = \langle guard_{ij}, weight_{ij}, EVT\_D_{ij} \rangle\}$ *where: i) k is the number of operands in CF Fi, ii) $weight_{ij}$ is the weight of the operand OP_{ij}, iii) EVT_D_{ij} are the events that are directly contained in an operand OP_{ij}.*

• finally we have defined several relations and functions that are required for the formalization of the precedence relations. For instance some functions permit to return some important informations in the tree : *i)* the function *ancestor* permits to associate to each operand all the operands where it is nested (its ancestor operands in the tree); *ii)* the function *brother* permits to identify the operands that belong to the same CF ALT. In a given tree the brother operands are the operands that belong to the same level and that have the same index *i*. Hence, the operands of the same sibling are not all necessarily brothers, since some of them came from lower level when built the tree. This function is used in the precedence relations to enforce that the events that belong to different operands of an ALT CF, or that have in their ancestor operands that are brothers, must not be ordered.

Illustration 1: in Fig.1, $ancestor[\{OP_{00}\}] = \emptyset$, and $ancestor[\{OP_{21}\}] = \{OP_{11}, OP_{00}\}$.
Illustration 2: in Fig.1, the operands OP_{21}, OP_{22} belong to the same CF ALT, thus they are brothers. $brother[\{OP_{21}\}] = \{OP_{22}\}$
The other functions and relations permit to return some informations about the operands.
In the sequel we resume some of them that are required in this paper and we define a new ones.

We use the following functions to manipulate the operands:

- EVT_D permits to get the events that are directly contained in each operand,
- EVT_G permits to get all the events that are contained in an operand including those which are contained in its nested operands.

In the following we need to define a function that is required later to deal with the evaluation of guards and the synchronization issues.

We assume that each operand of a CF has only one first event. Intuitively, a first event is an event that has not a preceding events in the considered operand. We define the function $first$ that permits to get the first event of each operand: $first : OP \rightarrow EVT$

first=$\{(X,e) | X \in OP \wedge e \in EVT\_G(X) \wedge (\forall e')[e' \in EVT \wedge e' <^*_{caus} e \Rightarrow e' \notin EVT\_G(X)]\}$

The causal semantics (Dhaou et al., 2017) is based on fourth precedence relations:

1. the synchronization relation $< Sync$ allows to each lifeline to synchronize its behaviour; it expresses that a given message m is received if it was sent previously,

2. the emission-emission relation $< EE$ permits to order two successive events emitted by the same lifeline,
3. the reception emission $< RE$ relation expresses that the reception of an event by a lifeline causes the emission of the event that is successive to it,
4. the hidden relations $< Hcaus$ are the relations between the events of LOOP CF of the current iteration and the events of the previous iterations. These relations appear when the LOOP operand is flatten at least one time. They are due to the overlapping between the occurrence of the events of different iterations of a loop CF.

In this paper, we propose to cover an important aspect of CF that should not be disregarded: it is the constraint interaction that guards CF. Indeed, the evaluation of guards constraints the occurrence of the events. There are two aspects to consider for the guards: their evaluations and the effects of their evaluations on the occurrence of the events of the same operand and the events of the other operands; this is the synchronization issue.

3 GUARD EVALUATION

All kind of CF can be guarded. There are several features that must be considered. They lie on how to, and who must, evaluate the guard, when and how many time the guard evaluation should be done.

The standard semantics states that the guard has to be evaluated before the occurrence of the first event of each operand. This definition is ambiguous. Moreover the first event of an operand was not clearly defined. We propose an approach for guard evaluation that is faithful with the standard semantics. In our approach we have defined the first event with some restrictions. We consider the guard evaluation as an internal operation that is not visible from the environment. Hence, we use fictitious events (τ events) to deal with guard evaluation as well as the synchronization issue. Moreover the fictitious events will ensure the synchronisation between the operands once the guard is evaluated.
For each operand, every fictitious event belongs to the same lifeline of the first event of the considered operand; it is placed above the first event. For each guarded operand, the guard is evaluated one time by the lifeline of its fictitious event. For an ALT CF, the guard of each operand is evaluated one time by the corresponding fictitious event. If we have, simultaneously, several operands with true guards and whose their fictitious events satisfy the precedence constraints, only one fictitious event will be chosen to occur in a non-deterministic way. This is guaranteed by our semantics since it is an interleaving[1] semantics.

In the sequel we provide the formal definition of the fictitious events as well as the precedence relations that are caused by their introduction in the SD.

3.1 Formal definition of fictitious events

For each guarded operand OP_{ij}, we opted for the use of two fictitious events (positive and negative) (as depicted in (Fig.3), since they have distinct execution effects. The positive fictitious event (τ_{ij}^{+}) occurs when the guard is evaluated to *True*, and permits to the events of the corresponding operand to occur; the negative fictitious event (τ_{ij}^{-}) occurs when the guard is evaluated to *False*, and permits to prohibit the occurrence of the events of the concerned operand. In the following, we define some sets and functions that are used in the sequel.

- Fic_pos and Fic_neg represent, respectively the sets of positive and negative fictitious events, such that $Fic = Fic\_pos \cup Fic\_neg$
- We define two partial bijective functions that permit to associate respectively to each operand its positive and negative fictitious events.
  - $Fct\_\tau\_pos : OP \rightarrowtail\!\!\!\!\!\!\rightarrow Fic\_pos$: for each operand we associate a positive fictitious event,
  - $Fct\_\tau\_neg : OP \rightarrowtail\!\!\!\!\!\!\rightarrow Fic\_neg$: for each operand we associate a negative fictitious event,
- $Fct\_l\tau : Fic \rightarrow L$ permits to associate to each fictitious event its correspondent lifeline. Each fictitious event belongs to the lifeline of the first event of the considered operand,
- $<_\tau$: denotes the partial order relationship that contains the precedence relationships that are caused by the addition of the fictitious events. This relation is detailed in the sequel.

3.2 The Precedence Relations Induced by the Introduction of Fictitious Events

The fictitious events are inserted between the first event of an operand and its preceding events. This causes new precedence relations (i.e new couples of events) between them and the events of the SD.

The precedence relation $<_\tau$ is decomposed into two precedence relations $<_{\tau 1}$ and $<_{\tau 2}$ that are

[1] i.e. two events may not occur at exactly the same time.

detailed in the following.

For an operand OP_{ij}, the fictitious events τ_{ij}^{+} and τ_{ij}^{-} are both located between the first event of OP_{ij} and all its preceding events. Consequently, they inherit all the preceding events of the first event of the considered operand. These new precedence relations are computed from a relation noted $<_{\tau 1}$.

Illustration: in Fig.3, according to the $<_{EE}$ relationship, the event $!m1$ precedes the first event $!m2$ of the LOOP operand ; by inserting the fictitious events ($\tau+$ and $\tau-$), to evaluate the guard of the LOOP operand, the event $!m1$ becomes the preceding event of both $\tau+$ and $\tau-$ events.

$$<_{\tau 1} = \{(e,e') | e \in EVT \wedge e' \in Fic \wedge (\exists X)[(X \in OP) \wedge (e' = Fct\_\tau\_pos(X) \vee e' = Fct\_\tau\_neg(X)) \wedge ((e, first(X)) \in <_{caus})]\}$$

Moreover, the positive fictitious event τ_{ij}^{+} becomes the only preceding event of the first event, this is computed from $<_{\tau 2}$ relation.
Illustration: in Fig. 3, τ^{+} becomes the only preceding event of the event $!m2$.

$$<_{\tau 2} = \{(e,e') | e \in Fic\_pos \wedge (\exists X) [X \in OP \wedge e = Fct\_\tau\_pos(X) \wedge (e' = first(X))]\}$$

To recapitulate, the relation $<_{\tau 2}$ permits to generate new precedence relations that must be added to $<_{caus}$ relationships, however the relations $<_{RE}, <_{EE}, <_{hcaus}$ are overloaded with $<_{\tau 1}$ relationship. Up to now, we define a new relation $<_{causG}$ as follows.

$$<_{causG} = <_{Sync} \cup <_{REG} \cup <_{EEG} \cup <_{hcausG} \cup <_{\tau 2}$$

The relations $<_{REG}$, $<_{EEG}$, $<_{hcausG}$ are obtained by overloading the $<_{RE}$, $<_{EE}$, $<_{hcaus}$ relations with the $<_{\tau 1}$ relation. That means the ordering of events depends on the cumulative rules of the relationships. The valid traces are those which can be generated satisfying these orders.

The definitions provided by the standard for the computation of traces of an SD are not formalized. The rules that we defined in this paper can be adjusted to formalize the definitions of the standard by restoring some constraints that we relaxed. In the same way, these rules can be adapted for any kind of semantics by strengthening or weakening some constraints. The ordering of events depends on the cumulative rules of the relationships. The valid traces are those which can be generated with respect to these orders.
The semantics defined in this paper assumes that all the behaviour is explicitly specified in the diagram. The behaviour that is not modelled is considered as invalid.

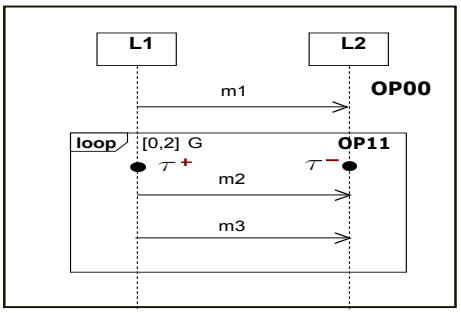

Figure 3: Fictitious Events

4 BEHAVIOUR OF SEQUENCE DIAGRAMS

The behaviour of a given SD is a set of traces. The trace is a set of events occurrences. The checking of the occurrence of all the preceding events for an event constraints its occurrence. However its occurrence depends on other constraints, such as the value of its state, that we need to define precisely, and the guard evaluation if it is about an event that belongs to guarded CF.

4.1 The state of an event

According to a run, an event which belongs to a basic SD can have two obvious basic states: occurred or not yet occurred. However, these basic states are not sufficient to express the state of an event in an SD with sophisticated structures (nested CF). Indeed, each event that belongs to such SD can be: either not yet occurred, or occurred or ignored one or several times (if it belongs to LOOP operand of nested CF that contains LOOP operand), or consumed. Then, the variable *state* is defined as follows.

$$state : EVT \rightarrow NAT$$

Depending on the precedence relations on events, we can define which event should be run according to their state. The initial value of the state of each event corresponds to its maximal value of occurrence in each run. The state of an event is decreased whenever it is occurred or ignored. To describe the state of an event e, we use the following vocabulary: *i)* not yet occurred, *ii)* occurred: if the event e is executed or ignored one or several times, *iii)* consumed: when the state of the event is equal to zero. During its execution, an SD can be in one state among the following

states: *i)* an initial state, when all its events are not yet occurred, *ii)* an intermediate state, *iii)* a final state, when all its events are consumed.
The notion of state is very important, indeed, it constraints the occurrence of a given event (for instance we decrement the state of an event whenever it is occurred, or if we want to prohibit its occurrence); it also serves to indicate the location of the considered event; this information is useful especially when we have several nested LOOP CF. This notion is also used for other purposes that we precise in the sequel.

4.2 The occurrence of events

An event occurs under some *trigger conditions* and produces some *execution effects*. The textual description of the definition of an event in a given SD with nested CF is done as follows.
The occurrence of a normal event: the trigger conditions consist in checking that:
i) the preceding events of the current event are occurred (executed or ignored),
ii) the current event is not yet consumed.
The occurrence of a fictitious events: The fictitious events (positive and negative) have additional executions effects, indeed we must add a trigger condition in which we check the value of guard of the considered operand.
The execution effects consist in:
i) updating the state of the event,
ii) updating the lifeline of the current event.

The fictitious events of an ALT or a LOOP CF have additional execution effects to deal with the synchronization issue that we explain in the sequel.

5 SYNCHRONIZATION ISSUE

For an ALT and a guarded LOOP CF, an important issue should be handled after the guard evaluation, is the synchronization between the lifelines covered by them. Indeed, the standard semantics of an ALT CF imposes that only one operand must be executed among several potential operands. When none guard is *True*, the CF is omitted. In the context of SD modelling behaviours of distributed system, since the lifelines involved in the interactions are independent, a synchronization issue arises. Therefore, in the context of distributed components, when the guard is evaluated, the decision must be propagated properly to the other lifelines to prevent a parallel execution of events that belongs to distinct operands, and the emergence of unspecified behaviours in the implementation.

Moreover, for the guarded LOOP operand, the events inside the operand are executed while the guard is evaluated to *True* and the maximal number of iterations is not reached. The LOOP CF represents a recursive application of the WEAK SEQUENCING operator, thus the traces that contain an overlapping between the occurrence of the events of different iterations are possible. Hence, once the guard become *False*, the events of the previous iterations that are not yet occurred must occur, while the events of the remaining iterations must not occur.

In the sequel we explain how the fictitious events are used to ensure the synchronization.

1. The execution effect of a positive fictitious event of a given operand of an ALT CF should: *i)* update its state, and its lifeline; *ii)* update the state of the negative fictitious event of the same operand for which it belongs, *iii)* decrease the state of each event of the brother operands in order to prohibit the occurrence of the events that are contained in them (i.e to ignore them),
2. The execution effect of a negative fictitious event of an ALT CF should decrease the state of all the events of the considered operand in order to ignore them.
3. The execution effect of a positive fictitious event of a LOOP CF should: *i)* update its state by decreasing it, *ii)* update its lifeline, *iii)* update the state of the of the negative fictitious event of the same operand,
4. The execution effect of a negative fictitious event of a LOOP operand should decrease the state of each event of the considered operand in the two following cases:

   i) if the guard is false from the beginning: none iteration of the LOOP operand is executed, hence we decrement the state of each event of the operand including the state of the events that belong to the nested operands of the considered operand (if they exist); *ii)* if the guard changes its value and becomes false during the iterations: the events of the current iteration must finish their occurrences (if we have some messages that are already sent, then their corresponding received events must occur), and the events of the remaining iterations must not occur.

The use of fictitious events causes new precedence relations that must be handled. However the fictitious events give us a high flexibility in defining execution strategies of the events. Moreover by ensuring the

tasks of evaluating guard and the synchronization between the lifelines, this permits us to define for the events of the SD a same shape (same trigger conditions and same execution effects).

6 RELATED WORK

The definitions that are given by the standard semantics (Group, 2015) concerning the guard evaluation are imprecise. This opens the door to the proposition of several approaches by the existing semantics that deal differently with this issue. However these approaches are not convenient for all kinds of systems. The existing semantics that are proposed in the literature for SD are of two categories (Zoltán and Hélène, 2011): the semantics of the first category are compositional ((Cengarle and Alexander, 2004), (Knapp and Wuttke, 2006), (øystein Haugen et al., 2005), (Harald, 2003), (Eichner et al., 2005), (Youcef, 2006), (Fernandes et al., 2007), (Shen et al., 2008)) such that they proceed to the parsing of the SD with CF into several basic SDs, then the semantics of an SD is mechanically built from the semantics of its constituent. The constituents are ordered and combined by the weak sequencing operator. In these semantics the guards are implicitly evaluated, hence several hidden issues are not treated explicitly.
In the semantics of the second category ((Cavarra and Filipe, 2004), (Juliana, 2006), (Harel and Maoz, 2008)), the SD is treated as a whole, and guards are explicitly evaluated. However some of them apply syntactic restrictions to avoid many inconsistencies. For instance, in the approach of (øystein Haugen et al., 2005), the guards are handled as local constraints. Contrarily to our approach their semantics allows non-local choices. However, it is not specified how to handle constraints if several guards are true simultaneously. In the works of (Cavarra and Filipe, 2004), (Juliana, 2006), (Shen et al., 2008), each lifeline can make its choice independently, however they impose an ordered guard evaluation of the operands (from top to bottom, and the first one evaluated as true be chosen). In (Eichner et al., 2005), the authors use a black box semantics to interpret CF, that consists to consider the CF as an atomic event, thus the behaviour of a CF does not interfere with the behaviours outside the CF. In some works, the authors adopt a non-standard interpretation of the ALT and the LOOP CF. In (Youcef, 2006), (Huszerl et al., 2008), (Cavarra and Filipe, 2004), (Maoz et al., 2011), the authors transform the ALT CF into an $IF-THEN-ELSE$ construct, hence there is no problem of synchronization between the lifelines; moreover they do not consider LOOP operand with guard that requires a special processing. In (Group, 2005), the authors define a deterministic ALT. In (Youcef, 2006), the authors use strict sequencing rather than weak for the LOOP CF to avoid a pathological case of divergence in LOOP combination when using asynchronous communication. Similarly, in (Knapp and Wuttke, 2006), the authors define a new CF SLOOP that enforces strict sequencing between the iterations. Hence a synchronization on entering and exiting CF is imposed. In (Juliana, 2006), the proposed approach considers a smaller number of CF (ALT, PAR, SEQ, and state invariants). The authors did not deal with synchronization issue and they did not consider the LOOP CF that requires a special processing for the definition of preceding events of each event. In our approach, we interpret the LOOP and ALT CF as in the standard semantics, differently of the interpretation in the structured programming languages. We deal with ALT CF having several true guards simultaneously and we deal with guarded loop CF. The guard evaluation is explicit. For an ALT CF the guard evaluation is done in one step to ensure that only one of the operands is chosen.

7 CONCLUSION

The causal semantics that we have proposed in our previous work (Dhaou et al., 2017), the meaning of SD with nested CF (ALT, OPT, LOOP) modelling behaviours of distributed systems is unambiguously interpreted using the partial order relations. The precedence relations allow to compute all possible valid traces for SDs with nested CF that model behaviours of distributed systems, by avoiding the parsing of SD equipped with (ALT, OPT, LOOP) CF, hence the compact syntactic representation is preserved. The proposed semantics can serves as basis for the derivation of traces of UML2.X SD, as well as for the definition of an operational semantics that facilitates the analysis of the SD.

In this paper we extended our semantics to cover an important aspect that is the guard evaluation and it consequences. This aspect is tackled with different manners by the existing semantics that are not usually appropriate in our context where the components are distributed. We are developing a plug-in for automatizing the computation of precedence relationships between the events of SD with nested CF. We plan to implement the semantics above in Event-B (Abrial, 1996), (Butler, 2008) to enable rigorous model analysis, the checking of the correctness of the model and some properties using the formal techniques of Event-B and its various tools Rodin: with a theorem-prover, and with ProB model-checker.

REFERENCES

Abrial, J.-R. (1996). *The B Book*. Cambridge University Press.

Aredo, D. B. (2002). A framework for semantics of uml sequence diagrams. *in PVS Journal of Universal Computer Science (JUCS)*, pages 674–697.

Butler, M. (2008). Incremental design of distributed systems with event-b. *Lecture Notes in Marktoberdorf Summer School*.

Cavarra, A. and Filipe, J. K. (2004). Formalizing liveness-enriched sequence diagrams using asms. In *Abstract State Machines 2004. Advances in Theory and Practice, 11th International Workshop, Lutherstadt Wittenberg, Germany, Proceedings*, pages 62–77.

Cengarle, M. V. and Alexander, K. (2004). Uml 2.0 interactions: semantics and refinement. *Technische Universitat München*, pages 85–99.

Cengarle, M. V., Graubmann, P., Wagner, S., and München, T. U. (2005). Semantics of uml 2.0 interactions with variabilities.

Dhaou, F., Mouakher, I., Attiogbé, C., and Bsaies, K. (2015). Extending causal semantics of uml2.0 sequence diagram for distributed systems. *Proceedings of the 10th International Conference on Software Engineering and Applications, Colmar, Alsace, France*, pages 339–347.

Dhaou, F., Mouakher, I., Attiogbé, C., and Bsaies, K. (2016). Refinement of UML2.0 sequence diagrams for distributed systems. In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) Lisbon, Portugal*, pages 310–318.

Dhaou, F., Mouakher, I., Attiogbé, C., and Bsaies, K. (2017). A causal semantics for UML2.0 sequence diagrams with nested combined fragments. pages 47–56.

Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., and Stehno, C. (2005). Compositional semantics for uml 2.0 sequence diagrams using petri nets. *Lecture Notes in Computer Science*, 3530:133148.

Fernandes, M., Tjell, S., Jorgensen, J. B., and Ribeiro, O. (2007). Designing tool support for translating use cases and uml 2.0 sequence diagrams into a coloured petri net. In *6th Int.Wsh. on Scenarios and State Machines*. IEEE Computer Society.

Grosu, R. and Smolka, S. (2005). Safety-liveness semantics for uml 2.0 sequence diagrams. In *5th Int. Conf. on Application of Concurrency to System Design*, pages 6–14.

Group, O. M. (2005). Omg unified modeling language (omg uml), superstructure version 2.5.

Group, O. M. (2015). Omg unified modeling language (omg uml), superstructure version 2.2.

Harald, S. (2003). Semantics of interactions in uml 2.0. In *HCC*, pages 129–136.

Harel, D. and Maoz, S. (2008). Assert and negate revisited: modal semantics for uml sequence diagrams. In *Software and System Modeling*, pages 237–253. John Wiley & Sons, Inc.

Huszerl, G., Waeselynck, H., and Zoltán Égel, András Kvi, Z. M. (2008). Refined design and testing framework, methodology and application results.

Juliana, K. F. (2006). Modelling concurrent interactions. *Theor. Comput. Sci.*, pages 203–220.

Knapp, A. and Wuttke, J. (2006). Model checking of uml 2.0 interactions. In Kühne, T., editor, *Models in Software Engineering*, pages 42–51. Springer.

Li, X., Liu, Z., , and He., J. (2004). A formal semantics of uml sequence diagram. *In Australian Software Engineering Conference IEEE Computer Society*, pages 168–177.

Maoz, S., Harel, D., and Kleinbort, A. (2011). A compiler for multimodal scenarios: transforming lscs into aspectj. *ACM Trans. Softw. Eng. Methodol.*, pages 18:1–18:41.

O.Tahir, C.Sibertin-Blanc, and J.Cardoso (2005). A causality-based semantics for uml sequence siagrams. In *23rd IASTED International Conference on Software Engineering*, pages 106–111. Acta Press.

øystein Haugen, Husa, K. E., Runde, R. K., and STAIRS (2005). Towards formal design with sequence diagrams. In *Software and System Modeling*, volume 4, pages 355–357. John Wiley & Sons, Inc.

Shen, H., Virani, A., and Niu, J. (2008). Formalize uml2 sequence diagrams. *11th IEEE High Assurance Systems Engineering Symposium, Nanjing, China*, pages 437–440.

Youcef, H. (2006). Branching time semantics for uml 2.0 sequence diagrams. *Lecture Notes in Computer Science: Formal Techniques for Networked and Distributed Systems*, pages 259–274.

Zoltán, M. and Hélène, W. (2011). The many meanings of uml2.0 sequence diagrams: a survey. *Software & Systems Modeling*, pages 489–514.