A Causal Semantics for UML2.0 Sequence Diagrams with Nested Combined Fragments

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

Keywords: UML2.0 Sequence Diagrams, Semantics, Nested Combined Fragments.

Abstract: Combined Fragments (CF) are the new features added to UML2.0 sequence diagrams (SD). They have widely increased its expressiveness power, permitting to model complex behaviours, they can be nested to allow more sophisticated behaviours. We focus on the most popular CF of control-flow ALT, OPT, LOOP, SEQ allowing to model respectively alternative, optional, iterative and sequential behaviours. They require a meticulous processing for the generation of partial order between their events. We proposed in a previous work, a causal semantics based on partial order theory, which is suitable for deriving of all possible valid traces for sequence diagrams with CF modelling behaviours of distributed systems. In this work, to deal with nested CF, we first update the formalization of sequence diagram, then we extend this semantics.

1 INTRODUCTION

Context. The speed of design, the intuition and the ease of graphical representation make UML2.0 sequence diagrams (SD) a privileged language often used by the engineers in the software industries. Although the Object Management Group (OMG) (Object Management Group, 2009) has defined an official standard semantics for UML2.0 SD, some shortcomings still persist. For instance, we report the inadequacy of the standard semantics for the computation of possible valid traces for an SD that models behaviours of a distributed system.

Motivation. The defined rules by the OMG for deriving partial order of a given basic SD impose to order the events along each lifeline, even if they are received from independent lifelines, which do not allow the computation of all possible valid behaviours. This lead to the emergence of unspecified behaviours in the implementation.

With UML2.0, the combined fragments allow the modelling of several kind of behaviours. We focus especially on a subcategory of CF: ALT, OPT, LOOP, SEQ; they permit a compact syntactic representation of behaviours. In contrast, they cause challenges for the determination of precedence relations between the events. To compute traces for SD equipped with these CF, the OMG standard recommends to flatten the underlying SD to obtain basic SDs that are semantically equivalent. However, the benefits of the compact syntactic representation are lost.

Moreover, the ALT and the LOOP CF have a different meaning than in the structured programming languages; although, to ease the processing of these CF, the existing approaches (Gàbor Huszerl, 2008), (Hammal, 2006), (Shen, 2013), restrict their use by interpreting them in the same way. However, in the standard they have much more flexible interpretations allowing to model more complex behaviours; for instance the ALT CF is not equivalent to the $IF-Then-Else$ structure, and in the LOOP CF, weak sequencing between the iterations is applied, rather than strict sequencing, permitting the interleaving of the occurrence of the events of different iterations.

In the practical cases, CF can be nested to model more sophisticated behaviours. All the cited problems are increasing. In the standard semantics, the notion of nested CF is briefly mentioned. In literature, few works (Shen, 2013), (Hammal, 2006), (Gàbor Huszerl, 2008) deal with nested CF. In (Gàbor Huszerl, 2008) the authors study the issues resulting of the nesting of some kinds of CF (different of those considered in this paper), and by limiting the nesting levels of CF (Gàbor Huszerl, 2008), (Hammal, 2006), or by proposing a complicated formalization very close to the target formalism (Shen, 2013).

Although the existing semantics that are proposed for UML2.0 SD are various (øystein Haugen and STAIRS, 2005), (Harel and Maoz, 2008), (Grosu and Smolka, 2005), (Cengarle et al., 2005), but they are usually based on the rules of the standard semantics

Dhaou, F., Mouakher, I., Attiogbé, J. and Bsaies, K.
A Causal Semantics for UML2.0 Sequence Diagrams with Nested Combined Fragments.
DOI: 10.5220/0006314100470056
In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017)*, pages 47-56
ISBN: 978-989-758-250-9

for the computation of traces of the SD, thus they are not suitable for SD modelling behaviours of distributed systems. This justifies the need of a semantics for UML2.0 SD with nested CF that models behaviours of distributed systems.

Contribution. This paper extends our previous work (Dhaou et al., 2015), in which we have extended an existing semantics proposed for UML 1.X (O.Tahir and J.Cardoso, 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.

We now propose some additional contributions that consist, in updating the formalization of sequence diagrams in order to support nested CF, as well as extending the previous results: we generalize the previous rules for the derivation of partial order between events.

Organization. The remainder of the article is structured as follows. In Section 2, we propose a running example that is used to illustrate our approach. Section 3 is devoted to an overview of the causal semantics. In Section 4, we provide the new formalization, based on set theory and tree structure, of UML2.0 SD that are equipped with nested CF. Then in Section 5, we explain our approach for the extension of causal semantics. Before concluding in Section 7, we present some related works in Section 6.

2 RUNNING EXAMPLE

We consider the interactions in a web site that proposes trainings as an example of UML2.0 SD modelling the behaviours of a distributed system (as depicted in Fig.1, Fig.2). The web site has independent components: the $training\_officer$, the *home_page*, the *custom_page*, the $training\_page$, and the *authentication_page*.

We model the update of the training catalog by the $training\_officer$ (as depicted in Fig.2). The $training\_officer$ has to authenticate at first (as depicted in Fig.1). He enters the web site url, he can be redirected directly to the *custom_page* if he has chosen, in the last connection, the option to remain connected for a limited duration. Otherwise, he is redirected to the *authentication_page*. We have three possible cases:

- a successful authentication,
- missing informations, in this case the site asks the user to complete them,
- wrong login and / or wrong password, in this case the site asks the training officer to correct them.

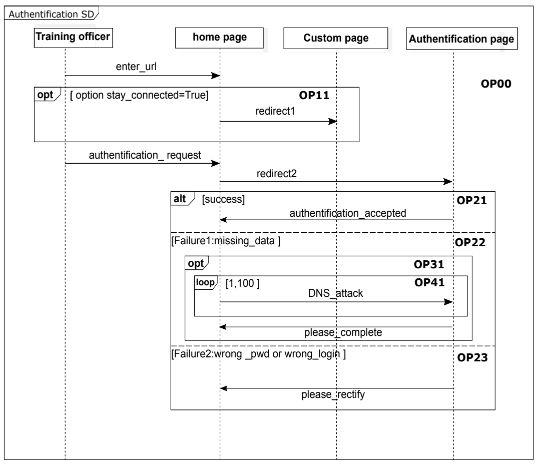

Figure 1: Authentification.

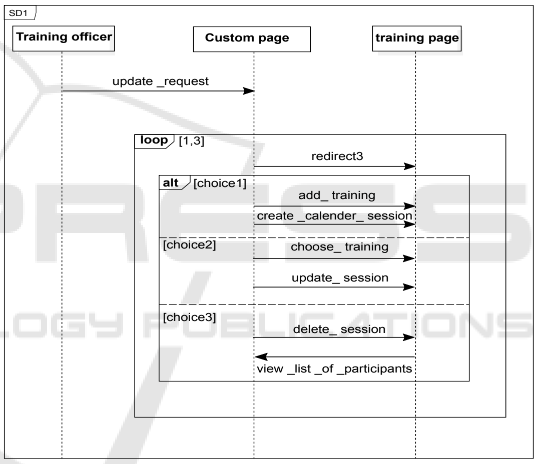

Figure 2: Update training catalog.

In the second case, domain name system (DNS) attacks may accidentally occur.

The Fig. 2 depicts the update of the training catalog. The training officer requests to update the training catalog, he is redirected to the $training\_page$ where he has three possible alternatives: *i)* he can add new training and create calender session; *ii)* he can choose training and update session, *iii)* he can remove training, the list of participant to this training is displayed.

3 CAUSAL SEMANTICS

The causal semantics was proposed (O.Tahir and J.Cardoso, 2005) for basic SD modelling behaviours of distributed systems. Its rules take into account the independence of the components, (modelled by lifelines), involved in the interactions. Indeed, in contrast with the standard semantics that totally order the events on each lifeline even for the receiving events from independent lifelines, the causal semantics imposes slighter scheduling constraints on the behaviour of lifelines results in more expressive SDs, since each SD describes a larger number of acceptable behaviours. This larger expressive power facilitates the task of the designer since a great number of cases have to be considered, and permits to prevent the issue of the emergence of unspecified behaviours in the implementation. The causal semantics is founded on a partial order theory. However, the causal semantics is mainly proposed for basics UML1.X SD modelling behaviours of distributed systems, and the application of its rules causes some inconsistencies (aberrant relations, deadlock and inadvertent triggers of some events (Dhaou et al., 2015)). Based on these verdicts, in our previous paper (Dhaou et al., 2015), we proposed an extension of the causal semantics that consists in defining and formalizing new rules for both relationships $<_{RE}$ and $<_{EE}$ for an SD with sequential ALT, OPT and LOOP CF by avoiding their flattening. The new defined relationships permit to obtain the partial order between events, that allows to compute all the possible valid traces of an SD modelling behaviours of a distributed system on a compact way. The previous approach consists in processing each concerning the various localizations of two events to be ordered.

Intuitively, the causal semantics (Sibertin-Blanc and J., 2005) is based on the idea of ordering events if there is a logical reason to do so. Two successive events that are sent by the same lifeline are necessarily ordered. Two successive events on the same lifeline that are received by two independent lifelines are not necessarily ordered; even if the messages are sent by the same lifeline, the receiving events are not ordered (if the architecture do not allow their ordering). In the following, we present briefly the rules of causal semantics as defined in (O.Tahir and J.Cardoso, 2005) in informal way.

Synchronization Relationship $<_{SYNC}$. Each message m is received only if it was sent previously.
Reception-Emission Relationship $<_{RE}$. Receiving a message causes the sending of the message that is directly consecutive to it.
Emission-Emission Relationship $<_{EE}$. If two messages are sent by the same lifeline their sending events are ordered.
Causal order Relation $<_{caus}$. This relation is defined as follows: $<_{caus} = (<_{SYNC} \bigcup <_{RE} \bigcup <_{EE})$

The transitive closure of the relation $<_{caus}$ that we note $<^{+}_{caus}$ permits to obtain all the causal dependencies between the events of the SD. The event occurrence depends on the partial order relationship $<_{caus}$.
Illustration: in the Fig. 1, we consider the messages $enter\_url, redirect1, authentication\_request$, according to the rules of the standard semantics: on the $training\_officer$ lifeline the events $!enter\_url$[1] and $!authentication\_request$ are ordered, on $home\_page$ lifeline the events $?enter\_url$[2], $!redirect1$ and $?authentication\_request$ are ordered. With the rules of the causal semantics the reception of the messages $enter\_url$ and $authentication\_request$ are not ordered thus we obtain much more traces than with the rules of the standard.

By applying the same approach for an SD with nested CF, we face a problem of combinatorial explosion in the number of possible cases for the localization of two successive events to order. In the next section, we propose a new formalization of the sequence diagrams that is required for the extension of causal semantics to support nested combined fragments.

4 TOWARD A NEW FORMALIZATION

We consider a sub-set of SD containing combined fragment of control-flow ALT, OPT, LOOP and SEQ CF. The considered CF are sequential, and can be nested to model more sophisticated behaviours. We assume that the operands of the CF do not overlap, but can be nested. For the formalization of sequence diagrams equipped with nested CF, we choose, on the one hand, the set theory notations[3] that is a privileged way due to its several advantages. For instance, although it is founded on first order logic, it permits to manipulate objects of high order such as sets and relations of any depth (that is, sets and relations built themselves on sets and relations, and so on) (Abrial, 1996). On the other hand, we use the tree structure that is hierarchic by nature and it is convenient to capture the nested structure of SD, and allow to represent them in an intuitive way.

4.1 Background

A tree is a data structure consisting of nodes organized as a hierarchy. We summarize in the following the vocabulary that relates to tree structure. A tree is either empty or a root node connected to 0 or more

[1] $!m$ denotes the sent of m message.
[2] $?m$ denotes the reception of m message.
[3] N.B we use the same set theory notation as those of Event-B method.

trees (called subtrees). *i)* Root: it is a node at the top of the tree. There is only one root per tree and one path from root node to any node, *ii)* parent: any node (except the root node) that has one edge downward to a node is called a parent, *iii)* child: node below a given node connected by its edge upward is called a child node, *iv)* each node is either a *leaf* or an *internal node*: an internal node has one or more children and a leaf node (external node) has no children, *v)* nodes with the same parent are *siblings*, *vi)* the descendants of a node n are all nodes reached from node n to the leaf nodes, *vii)* a path is a sequence of nodes $n_0, n_1, ..., n_n$, where there is an edge from one node to an unique node. The path can be only downward, and it connect a node with a descendant, *viii)* the length of a path is the number of edges in the path, *ix)* the ancestors of a node n are all nodes found on the path from the root to node n, *x)* the depth of a node n is the *length* of the path from root to n.

4.2 Sequence Diagram Definitions

Definition 1. ***(Sequence Diagram)*** *A sequence diagram SD is a tuple*
$SD : \langle L, M, EVT, FCT\_s, FCT\_r, FCT\_l, OP, F, <_{caus}, tree\_OP \rangle$
where:

- *L is a not empty set of lifelines, and $card(L) \geq 2$,*
- *M is a not empty set of asynchronous messages which is well formed. The set M is well formed if every message is identified by a pair of events: a sent event and a received event,*
- *$EVT = E\_s \cup E\_r$ is a set of events such that $card(EVT) \geq 2$[4], $E\_s = \{!m \mid m \in M\}$ and $E\_r = \{?m \mid m \in M\}$ denote respectively the set of sent events and the set of received events such that $E\_s \cap E\_r = \emptyset$,*
- *for a set of message M we define two bijective functions: i) $FCT\_s : M \rightarrowtail\!\!\!\twoheadrightarrow E\_s$[5]: for each message we associate one sent event; ii) $FCT\_r : M \rightarrowtail\!\!\!\twoheadrightarrow E\_r$: for each message we associate one received event,*
- *$FCT\_l : EVT \twoheadrightarrow L$[6] a total surjective function that associates to each event one lifeline, the transmitter or the receiver,*
- *$F = \{F_1, F_2, ..., F_n\}$ is the set of n combined fragments, where $F_i = \langle OP_i, operator_i, L_i \rangle$ is a CF that is identified by its operands, an operator, and the set of lifelines that are covered by it,*
- *$<_{caus} \subseteq EVT \leftrightarrow EVT$ denotes the partial order relationship,*
- *OP: a set of operands,*
- *$tree\_OP$ is a partial function that allows to structure the SD in the form of a tree of operands.*

To obtain the local order within each lifeline noted $<_{SD,l}$, we project the causal order relation $<^{+}_{caus}$ [7] on the lifeline l.

4.3 Operands and Tree Structure

Nesting combined fragments are not expressed in the meta-model, we propose its extension as illustrated in Fig.3. We redefine the class *Interaction Operand* as an abstract class that can be instantiated by: a leaf operand and nested operands, where this latter can be related to child operands.

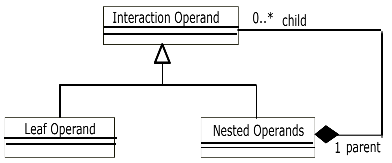

Figure 3: Interaction Operand Metamodel.

We propose a formalization of UML2.0 SD that is compliant with the standard UML2.0 meta-model up to a renaming of some constituents. We opted for the use of a tree structure that is hierarchic by nature, and that is convenient to capture the nested structure of SD allowing to represent them in an intuitive way. An SD is abstracted as a tree of operands. Intuitively combined fragment will be viewed as an operator together with its operands; this will be detailed in the sequel. Our approach supports multiple nesting levels of fragments. Neither the number of fragments enclosed by another fragment nor the depth of operand nesting are limited.

4.3.1 Operands

The SD is considered as a set of operands. We associate a label to each operand. Two operands with the same index i belong to the same combined fragment; for instance, in Fig. 1, OP_{21}, OP_{22} and OP_{23} belong to the same CF ALT.

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. 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. We assume that each operand of a

[4]Cardinal of a set E.

[5]$\rightarrowtail\!\!\!\twoheadrightarrow$ denotes a bijective function.

[6]$\twoheadrightarrow$ denotes a total surjection.

[7]R^+: the transitive closure of R.

CF has only one first event. The general definition of an operand in a combined fragment is given as follows:

Definition 2. ***(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}.

We use the following functions to manipulate the operands:

- EVT_D permits to get the events that are directly contained in each operand:
$$EVT\_D : OP \rightarrow \mathbb{P}(EVT)^{8}$$
- EVT_G permits to get all the events that are contained in an operand including those which are contained in its nested operands:
$$EVT\_G : OP \rightarrow \mathbb{P}(EVT)$$
- *weight* permits to return the weight of each operand:
$$weight : OP \rightarrow NAT^{+}$$

The instantiation of the definition 2 for SEQ, ALT, OPT and LOOP CF is given as follows:

Definition 3. ***(Operand in the SEQ Combined Fragment)*** *A sequential combined fragment contains only one operand linked to the SEQ operator.*

$$OP_i^{\text{SEQ}} = \{OP_{i1}\}$$

where $OP_{i1} = \langle True, 1, EVT\_D_{i1} \rangle$

Definition 4. ***(Operand in the ALT Combined Fragment)*** *An alternative combined fragment F_i is composed of a set of k operands:*

$$OP_i^{\text{ALT}} = \{OP_{i1}, ..., OP_{ik}\}$$

where $OP_{ij} = \langle (g_{ij} \wedge A_{ij}), 1, EVT\_D_{ij} \rangle$ *and g_{ij} is the guard of the operand OP_{ij}, the weight is equal to 1, and A_{ij} is a predicate that ensures that we have exclusively one operand that must occur in combined fragment F_i.*

Definition 5. ***(Operand in the OPT Combined Fragment)*** *An optional combined fragment F_i is composed of one operand:*

$$OP_i^{\text{OPT}} = \{OP_{i1}\}$$

where $OP_{i1} = \langle g_{ij}, 1, EVT\_D_{ij} \rangle$ *and g_{i1} is the guard of the operand OP_{ij}.*

[8]$\mathbb{P}(EVT)$: set of subsets E.

$$\begin{tikzcd}
& OP_{00} & & \\
OP_{11} \arrow[ur] & OP_{21} \arrow[u] & OP_{22} \arrow[ul] & OP_{23} \arrow[ull] \\
& & OP_{31} \arrow[u] & \\
& & OP_{41} \arrow[u] &
\end{tikzcd}$$

Figure 4: Tree associated to the SD of Fig.1

Definition 6. ***(Operand in the LOOP Combined Fragment)*** *An iterative CF has one operand. Its events will execute for at least the minimum min_i of iterations and no more than the maximum max_i of iterations as long as the guard is evaluated to True.*

$$OP_i^{\text{LOOP}} = \{OP_{i1}\}$$

where $OP_{i1} = \langle ((g_{i1} \wedge B_i) \vee C_i), max_i, EVT\_D_{i1} \rangle$ *and g_{i1} is the guard of the* LOOP *operand; B_i is a predicate which indicates that the current iteration is between min_i and max_i values of iteration. C_i is a predicate that is satisfied if the current iteration is less than min_i.*

4.3.2 Tree Structure

An SD is a tree that is composed by a set of linked operands, such that each operand has at maximum one direct ancestor. For instance, the Fig. 4 illustrates the associated tree for the SD of the Fig. 1. The root operand OP_{00} has no ancestor. Each node represents an operand, and has as children the nested operands that are inside it.

We define tree structure for SD operands as follows:

Definition 7. ***(Tree Structure for SD Operands)*** *The tree structure $tree\_OP$ related to an SD is defined as a partial function:*

$$tree\_OP : OP \nrightarrow OP$$

which is acyclic and non-reflexive. The root is the only operand that does not have a parent:

$$(\forall X)[X \in OP \wedge X \notin dom(tree\_OP) \wedge X \in ran(tree\_OP) \Rightarrow X = OP_{00}]$$

We introduce new relations *ancestor* and *brother* such that the first one permits to associate to each operand all the operands where it is nested (its node ancestors in the $tree\_OP$); the second one permits to link each operand to the other operands of the same combined fragment (its brothers).

– *ancestor*: a binary transitive relation that is defined on *OP*.

$$ancestor : OP \leftrightarrow OP^{9}$$

For an operand X we compute its ancestors as follows:

$$ancestor[\{X\}]^{10} = \bigcup_{s \in \{1,..,d\}} \{tree\_OP^{s}(X)\}$$

Where d is the depth of the node X in the $tree\_OP$.
Illustration: $ancestor[\{OP_{00}\}] = \emptyset$, and $ancestor[\{OP_{31}\}] = \{OP_{21}, OP_{00}\}$.

– *brother*: a binary transitive relation that is defined on a set *OP*.

$$brother : OP \leftrightarrow OP$$

Two operands are brothers if they belong to the same CF (it's the case of operands of an ALT CF).

$$\textbf{brother} = \{(OP_{ij}, OP_{tk}) | (OP_{ij}, OP_{tk}) \in OP^{2} \wedge (i = t \wedge j \neq k))\}$$

Illustration: the operands $OP_{22}, OP_{21}, OP_{23}$ belong to the same CF ALT, thus they are brothers.
$brother[\{OP_{11}\}] = \emptyset$ and
$brother[\{OP_{22}\}] = \{OP_{21}, OP_{23}\}$

4.3.3 Weight of an Event

We overload the function *weight* to associate the weight of the path between two operands.

$$weight : (OP \times OP) \rightarrow NAT^{+}$$

We overload the function *weight* that permits to associate to each event its maximal number of occurrence.

$$weight : EVT \rightarrow NAT^{+}$$

For an event *evt* such that $evt \in EVT\_D(X)$, we compute its weight as follows.

$$weight(evt) = \prod_{s \in \{0,d\}} weight(tree\_OP^{s}(X)) = weight(OP_{00}, X).\ With\ d = depth\ of\ X$$

These formalizations of the structures SD and CF are used as a basis for the extension of the causal relationships that permits to compute the partial order between the events of the SD.

[9] $\leftrightarrow$ denotes a relation.
[10] $R[\{e\}]$: Relational image; gives the set of images.

5 EXTENSION OF THE CAUSAL SEMANTICS

The choice we make for structuring the sequence diagram in the form of tree gives us the advantage of redefining the $<_{RE}$ and $<_{EE}$ relationships, to support nested CF, by optimizing the number of possible cases of event localizations to order. The synchronisation the $<_{SYNC}$ relationship is unchangeable. The formalizations of $<_{RE}$ and $<_{EE}$ relationships permit to order two events that belong to the same lifeline and that are successive.

To detail a bit, and to alleviate the presentation of the formalization of $<_{RE}$ and $<_{EE}$ relationships, we introduce three binary relations $not\_in\_brother$, $succ1$ and $succ2$. In the following, we first give the intuition of each of them before their formalizations.

Two successive events that belong to distinct operands of an ALT CF must not be ordered.

The relation $not\_in\_brother$ expresses this intuition: the successive events of an ALT CF to be ordered should not belong to the same ancestor at any level (or brother operands).

$$\textbf{not\_in\_brother}=\{(e,e') | (e,e') \in EVT^{2} \wedge (\forall X)(\forall Y) [X \in (ancestor[\{EVT\_D^{-1}(e)\}] \cup \{EVT\_D^{-1}(e)\}) \wedge Y \in (ancestor[\{EVT\_D^{-1}(e')\}] \cup \{EVT\_D^{-1}(e')\}) \Rightarrow (X,Y) \notin brother]\}$$

Illustration:
in Fig.1, the event$!please_complete \in OP22$, the event $!please\_rectify \in OP23$ and $OP22 \in brother[\{OP23\}]$, hence the events $!please\_complete$ and $!please\_rectify$ should not be ordered.
The event $!DNS\_attacks \in OP41$, the event $authentication\_accepted \in OP21$, $ancestor[\{OP23\}] = OP22$ and $OP22 \in brother[\{OP21\}]$, hence the events $authentication\_accepted$ and $!DNS\_attacks$ should not be ordered.

Formally, we distinguish two successive events in different ways with two distinct relations $succ1$ and $succ2$. These relations are used respectively in the formalization of $<_{EE}$ and $<_{RE}$ relationships. The relation $succ1$ relates two events that belong to the same lifeline and which are successive. Nevertheless, we admit between them, events that must necessarily belong to an operand that can be omitted (i.e. the events between successive events do not belong to any operand ancestor of the operands of the considered events).

succ1=$\{(e,e')|(e,e') \in EVT^2 \wedge (\exists l)[l \in L \wedge e <^*_{SD,l} e' \wedge (\forall e")[e" \in EVT \wedge (e <^*_{SD,l} e" \wedge e" <^*_{SD,l} e') \Rightarrow EVT\_D^{-1}(e") \notin (ancestor[\{EVT\_D^{-1}(e)\}] \cup ancestor[\{EVT\_D^{-1}(e')\}])]]\}$

The relation $succ2$ expresses the same conditions and effects as those defined in $succ1$ relationships, moreover it expresses that we admit received events between the successive events.

succ2= $\{(e,e')|(e,e') \in EVT^2 \wedge (\exists l)[l \in L \wedge e <^*_{SD,l} e' \wedge (\forall e")[e" \in EVT \wedge (e <^*_{SD,l} e" \wedge e" <^*_{SD,l} e') \Rightarrow e" \in ran(FCT\_r) \vee EVT\_D^{-1}(e") \notin (ancestor[\{EVT\_D^{-1}(e)\}] \cup ancestor[\{EVT\_D^{-1}(e')\}])]]\}$

The relationship $<_{EE}$ permits to order two sent events that satisfy the conditions expressed in $not\_in\_brother$ and $succ1$ relations.

$<_{\mathbf{EE}}= \{(e,e')|[(e,e') \in (EVT)^2 \wedge e \in ran(FCT\_s) \wedge e' \in ran(FCT\_s) \wedge (e,e') \in not\_in\_brother \wedge (e,e') \in succ1\}]$

The relationship $<_{RE}$ permits to order two events such that the first one is a received event and the second one is a sent event, and both of them satisfy the conditions expressed in $not\_in\_brother$ and $succ2$ relations.

$<_{\mathbf{RE}}= \{(e,e')|[(e,e') \in (EVT)^2 \wedge e \in ran(FCT\_r) \wedge e' \in ran(FCT\_s) \wedge (e,e') \in not\_in\_brother \wedge (e,e') \in succ2\}]$

5.1 Hidden Precedence Relations in LOOP Combined Fragment

The events inside a LOOP operand can have as preceding events that can be located:

- for the first iteration: *i)* either outside the LOOP operand and/or, *ii)* inside the LOOP operand of the same iteration.
- from the second iteration: *i)* either outside the LOOP operand and/or, *ii)* inside the LOOP operand of the same iteration and/or of the previous iterations.

We call hidden relations the relations between the events of LOOP operand of the current iteration and the events of the previous iterations (Fig.5). These relations appear when the LOOP operand is flatten at least one time. Hence, the necessity of defining a new relation $<_{hcaus}$ in which we express the constraints of precedence between the events of the current iteration and the events of the previous iteration. In order to compute the hidden precedence relations, we propose the following steps: we flatten the LOOP operand only once whatever is the number of iterations; we obtain an intermediate sequence diagram SD'.

In SD', we rename the operands as well as the events of the second iteration with the same name as those of the preceding iteration by labelling them with a single quote (Fig. 6). We define the set EVT' to represent the events of the next iteration. $<'_{RE}$ and $<'_{EE}$ are respectively the reception-emission, and the emission-emission relationships associated to the SD'.

In an SD we can have several LOOP operand that can be sequenced or nested. In this case, the same processing is applied by computing for each LOOP operand its hidden relationships; we note $<_{hcausX}$, the hidden relations of a given LOOP operand named X. The formalization of the hidden relationships for a LOOP operand X is given as follows.

$$<_{\mathbf{hcausX}}= \{(e,e')|e \in EVT \wedge e' \in EVT' \wedge (e,e') \in <'_{RE} \vee (e,e') \in <'_{EE}\}$$

Illustration1: consider the SD' (depicted in Fig. 5), in the first iteration, the preceding event of $!m2$ is $!m1$, the preceding events of $!m3$ are $?m1$ and $?m2$. In the second iteration, the preceding event of $!m2'$ is $!m4$, the preceding events of $!m3'$ are $?m4$ and $?m2'$.

Illustration2: as we aforementioned, for an ALT CF, only one operand must be executed, hence it is intuitive that the events that belong to distinct operands must not be ordered, otherwise we have the deadlocks of some events.
However, in some particular cases of nested structure, we can face a problem that the events of distinct operands of the same ALT CF (brother operands) can have precedence relations. The Fig. 7 represents a possible execution of the SD (depicted in Fig.6) containing nested CF. In the first iteration of the LOOP CF, the first operand of the ALT CF is executed; in the second iteration of the LOOP CF, the third operand of the ALT CF is executed. The event $!view\_list\_of\_participants$, that belongs to the third operand of the ALT CF, has among its preceding events: *i)* the event $?delete\_session$ that belongs to the same operand, *ii)* the events $?create\_calender\_session, ?add\_training$ which both belong to the first operand of the the ALT CF. Which is problematic, since the events of brother operands are not ordered. This justifies the renaming of the events and the operands of the next iteration to prevent this problem.

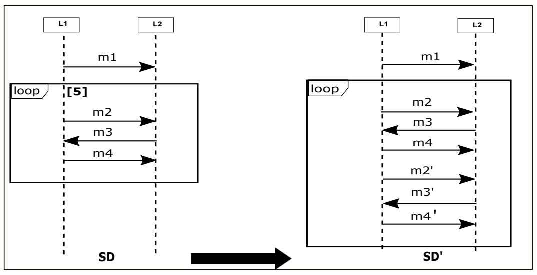

Figure 5: Processing of an SD with LOOP CF.

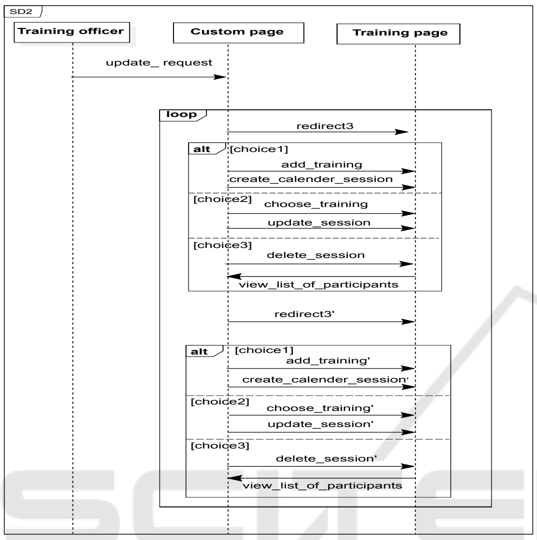

Figure 6: Processing of an SD with Nested CF.

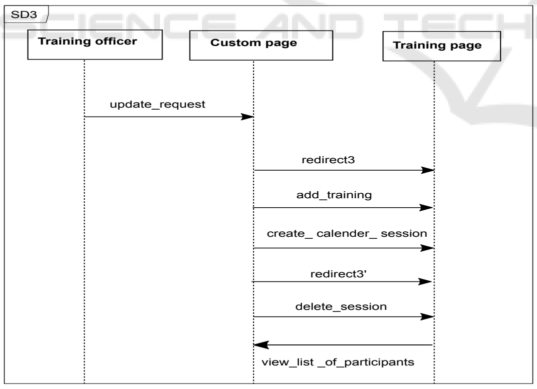

Figure 7: Possible execution of The SD of the Fig. 6.

In an SD we can have several LOOP operand that can be sequenced or nested. In this case, the same processing is applied by computing for each LOOP operand its hidden relationships; the entire hidden relation is the union of the hidden relations of each LOOP operand. Now, the causal relationships is computed as follows.

$$<_{caus} = <_{SYNC} \cup <_{RE} \cup <_{EE} \cup <_{hcaus}$$

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 defined rules ($<_{RE}$, $<_{EE}$, $<_{hcaus}$) may be applied to the standard semantics by restoring the constraints that we relaxed. In the same way, these rules can be adapted for any kind of semantics by strengthening or weakening some constraints.

5.2 Behaviour of Sequence Diagrams

The behaviour of a given SD is a set of traces. The trace is a set of events occurrences. The occurrence of an event depends on its definition.

5.2.1 The State of an Event

The causal semantics (O.Tahir and J.Cardoso, 2005) deals only with basic SD. In the standard semantics, even the UML2.0 SD, that are equipped with CF, are treated as basic SD, since the standard recommends their flattening. An event which belongs to a basic SD can have two obvious basic states: executed or not yet executed. In our approach, we support sequence diagrams with sequential CF that can be nested. The basic states are not sufficient to express the state of an event in an SD with sophisticated structures (nested CF). Indeed, each event in such SD can be: not yet occurred, occurred, consumed one or several times. Then, the variable *state* is defined as follows.

$$state : EVT \to NAT$$

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:

- not yet occurred: when $state(e) = weight(e)$,
- occurred: if the event e is executed or ignored one or several times and $0 < state(e) < weight(e)$,
- consumed: when $state(e) = 0$.

During its execution, an SD can be in one state among the following states:

- an initial state, when all its events are not yet occurred,
- an intermediate state,
- a final state, when all its events are consumed: $state = EVT \times \{0\}$.

5.2.2 The Definition of an Event

The behaviour of an SD is the occurrence of its events. An event occurs under some conditions and produces some effects. Each event which belongs to SD with combined fragment has triggers conditions including causality with other events, require formalizing the computation of events preceding the current event, and execution effects.

The textual description of the definition an event in a given SD with nested CF is done as follows.

Triggers conditions consist in checking that: *i)* the precedents events of the current event are occurred (executed or ignored), *ii)* the current event is not yet consumed.

Execution effects consist in: *i)* updating the state of the event.

6 RELATED WORKS

Most of existing semantics that are proposed for UML2.0 SD are based on the definitions of the standard semantics for the computation of traces of an SD, thus they are not suitable for SD modelling behaviours of distributed systems.

This mismatch has motivated the work of (Sibertin-Blanc and J., 2005), (O.Tahir and J.Cardoso, 2005) who proposed a causal semantics for basic SD. In (Sibertin-Blanc and Tahir, 2006), to show the benefits and the adequacy of causal semantics for SD modelling distributed systems, the authors present three kinds of semantics: emission semantics, linear semantics and message sequence charts (MSC) semantics.

The linear semantics and the emission semantics impose a restrictive rules for the ordering of the events of the SD, and they are practicable only for a smaller kind of SD modelling some kind of systems. The linear semantics (Cardoso and Sibertin-Blanc, 2001) permits to order all the messages of the considered SD by imposing a total serialization of the emission and the reception of the messages. The emission semantics (Cardoso, Janette and Sibertin-Blanc, Christophe, 2002) imposes the total serialization of emission of all the messages. The semantics of the MSC (Rudolph et al., 1996), predecessors of the SD, suffers from ambiguity of semantics interpretation (Alur et al., 1996). In the authors show that it can exists some kind of partial order between events that depend strictly on the considered architecture (for instance, existence of one or several FIFO queue for all the process...). The different partial order defined are *(i)* visual order: it corresponds to the order of the events as they appear in the SD and that does not reflect the occurrence order of the events during the execution of the system, *(ii)* enforced order: it corresponds to the order of the events that the underlying architecture can guarantee to occur only in the specified order, and *(iii)* inferred order: it is defined as the transitive closure of enforced order.

In our work, we consider the causal semantics that is founded on a partial order theory. The causal-based order relation defined accounts for the precedence relation between the events of an SD as far as all the synchronization between lifelines are ensured by the exchanged events, without the use of a global controller or the addition of other messages.

The presence of CF causes difficulties for the derivation of traces, to ease the processing of the LOOP and the ALT CF, some works interpret these CF similarly as in structured programming languages which lead to the lose of some possible traces. In (Hammal, 2006), (Gàbor Huszerl, 2008), in order to reduce the non-determinism of the ALT CF, the authors transform it into an $IF - THEN - ELSE$ construct. In (Hammal, 2006), the authors state clearly that for the loop CF, they ''prefer use strict sequencing rather than weak one 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. In our approach, we interpret the LOOP and ALT CF as in the standard semantics, differently of the interpretation in the structured programming languages. Moreover, since the rules of causal semantics relaxed the constraint of the ordering of the events on each lifeline, we obtain all possible traces for an SD modelling behaviours of distributed systems.

Usually, the existing works neglect the issues related to the use of nested CF in an SD. Although their use allows several complex scenarios that can be aggregated in a single sequence diagram, an inherent difficulty of the interpretation and the analysis of the SD arises. The standard semantics (Object Management Group, 2009) does not propose a meta-model for nested CF. In the work of (Shen, 2013), the authors propose a formal semantics based on the rules of the standard semantics, and that is close to the target formalism (LTL) and complicated to apprehend. Moreover, they consider UML2.0 SDs having finite traces with LTL formalism, that represents basically infinite traces.

In (Gàbor Huszerl, 2008), the authors consider conformance operators (ASSERT, IGNORE, CONSIDER and NEGATE, permitting to categorize the traces as invalid or valid, ALT and PAR) CF, and they aim to define rules that allow or prohibit the nesting of these operators, by limiting the nesting level of CF to two levels.

In (Hammal, 2006), the authors propose a formal definitions for UML2.0 SD based on branching time semantics and partial orders in a denotational style, nevertheless the notion of nested CF is briefly mentioned, and no explicit formalization for them is proposed.

7 CONCLUSION

In this paper, we first defined an intuitive formalization, based on set theory and tree structure, of sequence diagrams equipped with combined fragments that can be nested. The formalization can be adapted for any kind of semantics. Then, we have extended the causal semantics by redefining its rules in a synthetic way. The new causal relationship allows to compute all possible valid traces for SDs with nested CF that model behaviours of distributed systems, by avoiding the flattening 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 UML2.0 SD, as well as for the definition of an operational semantics that facilitates the analysis of the SD. Our approach is not related to any target translation formalism and can be implemented by any formal method for its verification, although it is already implemented, (Dhaou et al., 2015), (Dhaou et al., 2016), by Event-B method offering powerful tools (Rodin/Pro-B).

Meanwhile, we are extending our proposal to cover other concepts like the gates and the state invariants which allow one to express more complex behaviours and to cover other CF, in particular those dedicated to model parallel behaviours and invalid behaviours. In addition, we currently study theoretical properties that are derived from the proposed semantics.

REFERENCES

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

Alur, R., Holzmann, G. J., and Peled, D. (1996). An Analyzer for Message Sequence Charts. In *SOFTWARE CONCEPTS AND TOOLS*, pages 304–313.

Cardoso, J. and Sibertin-Blanc, C. (2001). Ordering Actions in Sequence Diagrams of UML. In *23rd International Conference on Information Technology Interfaces*, pages 3–14, J. Marohnica bb, 10000 Zagreb, Croatia. University Computing Centre Uniersity of Zagreb.

Cardoso, Janette and Sibertin-Blanc, Christophe (2002). An Operational Semantics for UML Interaction: Sequencing of Actions and Local Control. *European Journal of Automatised Systems, APII-JESA*, 36(7):1015–1028.

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. *ICSOFT-EA 2015 - Proceedings of the 10th International Conference on Software Engineering and Applications, Colmar, Alsace, France*, pages 339–347.

Dhaou, F., Mouakher, I., Attiogbé, C., and Bsaïes, K. (2016). Refinement of UML2.0 Sequence Diagrams for Distributed Systems. In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA, Lisbon, Portugal, July 24 - 26, 2016.*, pages 310–318.

Gàbor Huszerl, Hélène Waeselynck (ed.), Z. E. A. K. Z. M. (2008). Refined Design and Testing Framework, Methodology and Application Results.

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*, page 614.

Hammal, Y. (2006). Branching Time Semantics for UML 2.0 Sequence Diagrams. *Lecture Notes in Computer Science: Formal Techniques for Networked and Distributed Systems - FORTE 2006*, pages 259–274.

Harel, D. and Maoz, S. (2008). Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. *Software and System Modeling*, 7(2):237–252.

Knapp, A. and Wuttke., J. (2006). Model Checking of UML 2.0 Interactions. In Khne, T., editor, *Models in Software Engineering*, pages 42–51. Springer.

Object Management Group (2009). OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2.

O.Tahir, C.-B. and J.Cardoso (2005). A Causality-Based Semantics for UML Sequence Diagrams. In *23rd IASTED International Conference on Software Engineering*, pages 106–111. Acta Press.

øystein Haugen, Knut Eilif Husa, R. K. R. and STAIRS (2005). Towards Formal Design with Sequence Diagrams. In *Software and System Modeling*, volume 4, pages 355–357. John Wiley & Sons, Inc.

Rudolph, E., Grabowski, J., and Graubmann, P. (1996). Tutorial on Message Sequence Charts (MSC'96).

Shen, H. (2013). *A Formal Framework for Analyzing Sequence Diagram*. PhD thesis.

Sibertin-Blanc, C. and Tahir, O. (2006). From UML1.x to UML 2.0 Semantics for Sequence Diagrams. In Ramos, F. F., Lrios, R. V., and Unger, H., editors, *IEEE International Symposium and School on Advance Distributed Systems (ISSADS), Mexico (Mexique)*. IEEE.

Sibertin-Blanc, C., T. O. and J., C. (2005). Interpretation of UML Sequence Diagrams as Causality Flows. In *Advanced Distributed Systems, 5th Int. School and Symposium (ISSAD)*, number 3563, pages 126–140. Acta Press.