Extending Causal Semantics of UML2.0 Sequence Diagram for Distributed Systems

Fatma Dhaou, Inès Mouakher, Christian Attiogbé, Khaled Bsaïes

Extending Causal Semantics of UML2.0 Sequence Diagram for Distributed Systems

Fatma Dhaou[1], Ines Mouakher[1], Christian Attiogbé [2] and Khaled Bsaies[1]
[1]*Faculty of Sciences of Tunis, Tunis, Tunisia*
[2]*University of Nantes, LINA UMR 6241, Nantes, France*

Keywords: Causal Semantics, Sequence Diagram UML2.0, Event-B.

Abstract: The imprecision of the definitions of UML2.0 sequence diagrams, given by the Object Management Group (OMG), does not allow the obtention of all the possible valid behaviours for a given distributed system, when communicating objects are independent. We choose the *causal semantics*, which is suitable for this kind of systems; we propose its extension to support complex behaviours, expressed with combined fragments. We propose the implementation of our approach with Event-B in order to check later on some properties of safety, liveness and fairness.

1 INTRODUCTION

Context. A meticulous development of distributed systems may begin by writing the scenarios which describe the most significant behaviours.

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. The long term goal of our work is the verification of consistency of refined SDs. The current state of the works related to UML2.0 semantics for distributed systems does not propose such a verification framework. The use of the formal method Event-B with the Rodin/ProB framework is motivated by the availability of tools allowing refinement checking.

The operational semantics, defined by OMG, for UML2.0 SD still suffer from some problems. One of these problems arises from their lack of precision, which represses their convenient use.

Motivation. The rules defined by the OMG for deriving partial order in a given SD, state that the events which belong to each lifeline must occur exactly in the same order, as they are specified, this is restrictive when the objects are independent, and do not allow the obtention of all possible valid behaviours of a distributed system. Moreover, the category of combined fragments (CF), like ALT, OPT and LOOP, used to represent compactly the SD must be flattened for the computation of traces; it is like if we handle basic SD, and we lose the utility of compact representation. The *weak sequencing* is applied for the obtention of traces, and the synchronisation point for the entry to, or for the exit from, a CF is not specified. These two points are considered as the source of the problem of *counterintuitive orderings* (Micskei and Waeselynck, 2011). We choose the causal semantics which is suitable for modelling distributed systems. And we extend its rules, since it was proposed only for basic SD of UML1.x. Consequently, the rules defined, in our approach, allow the obtention of intuitive traces. In fact, the events of SD are ordered, only, if there is a logical reason to do so. The causal semantics is not equipped with tools for its implementation. Thus, the use of formal method offering tools seems a necessity in our work. There is some formal frameworks (ex. Input/Output automata) supporting the refinement, which can be used as target formalism for the implementation. However, there is a very limited support for this approach. The powerful tools supporting the refinement, the similarity between both formalisms, SD UML2.0 and Event-B, making a translation straightforward justify our choice of the Event-B formalism.

Contribution. Our first contribution is the extension of causal semantics for SD UML2.0 equipped with CF (ALT, OPT and LOOP). The second contribution is the implementation in Event-B of SD equipped with extended causal semantics. Accordingly, we can use the resulting B-machines for refinement checking purpose, and for verifying some properties that may be either intrinsic to SD or related to the considered

Dhaou F., Mouakher I., Attiogbé C. and Bsaies K..
Extending Causal Semantics of UML2.0 Sequence Diagram for Distributed Systems.
DOI: 10.5220/0005517703390347
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 339-347
ISBN: 978-989-758-114-4

system.

Organisation. The remainder of the article is structured as follows. Section 2 is devoted to the UML2.0 semantics and causal semantics; their insufficiencies are underlined. Then, in Section 3, we propose the extension of causal semantics for the SD of UML2.0 equipped with the most popular CF like ALT, OPT and LOOP. Section 4 is dedicated to the implementation of SD that are equipped by extended causal semantics with Event-B approach. In Section 5, we provide an overview on the methodology that allows us to verify the refinement between two sequence diagrams. Before concluding in Section 7, we present some related works in Section 6.

2 UML2.0 SEMANTICS, CAUSAL SEMANTICS AND INSUFFICIENCIES

2.1 UML2.0 Semantics

According to the OMG, the UML2.0 semantics proposed for a basic SD imposes that the events of the same lifeline are ordered even if they are received from different lifelines, and synchronise the events of each message. More constraints may be added to establish a general ordering, and to restrict the set of possible traces. A given message m is identified by a pair of events $(!m, ?m)$, where $!m, ?m$ denotes respectively send and receive event[1]. The trace of SD is a sequence of event occurrences, and it is obtained by satisfying the order imposed by a given semantics.

In this paper, we focus on the ALT, OPT and LOOP CFs allowing to model complex behaviours of the considered system compactly. A CF has an interaction constraint (CI), it is a boolean expression which guards an operand, and it can be explicit or implicit. The ALT CF indicates a choice of behaviours. It allows to model two or more possible behaviours. However, neither the exact time nor the number of times of the evaluation of the guard are indicated. Moreover, when several CI are evaluated to true, at most one of the operands will be chosen. However, the OMG standard does not specify exactly which operand will be chosen. The OPT fragment is equivalent to an alternative fragment with an empty ELSE operand. The LOOP CF allows to model a set of iterations running in a loop. The CI may include a lower and an upper number of iterations of the loop as well as a boolean expression.

Insufficiencies. The traces obtained by the rules defined by the standard UML2.0 semantics of SD do not reflect the behaviours which can be produced by a distributed system during its execution, this can be questionable. Having a semantics which provides the possible valid behaviours of a given SD is crucial to analyse the properties of safety. We choose the causal semantics, which is an appropriate SD semantics basis for modeling distributed systems, and we extend it to support SD with the CF ALT, OPT and LOOP without flattening them.

2.2 Causal Semantics

With the causal semantics (Sibertin-Blanc et al., 2005), the scheduling of sent and received events is well-defined and is suitable for diagrams modelling distributed systems. The causal semantics is mainly based on three relations: *1)* **Synchronisation relation** $<_{sync}$. This relation allows each lifeline to synchronize its behaviour; it expresses that a given message m is received if it was sent previously. *2)* **Reception-Emission relation** $<_{RE}$. Receiving a message causes the sending of the message directly consecutive to it. *3)* **Emission-Emission relation** $<_{EE}$. If two messages are sent by the same lifeline, their sending events are ordered. In some particular cases of distributed system, if these messages are addressed to the same lifeline, their received events are also ordered. The global order relation $<^*_G$ is defined as the transitive closure of the three relations $<^*_G = <_{SYNC} \cup <_{RE} \cup <_{EE}$. To obtain a local order inside a lifeline o, noted $<_o$, we project the global order $<_{G^*}$ on the lifeline o. In the following, we show that the relations of causal semantics are not sufficient to support SD with CF.

Insufficiencies. The causal semantics (Tahir et al., 2005) was proposed only for SD UML1.x. The application of its relations for SD with CF generates some anomalies such as generation of aberrant relations (aberrant relations), deadlocks events (deadlocks events). These two anomalies are caused by relations which must not exist between events (example of two events which do not belong to the same operand of the ALT CF). Another anomaly is the inadvertent triggering. It is the case of an event following a CF LOOP, that can be triggered after one iteration of its preceding event inside the CF; normally, the triggering must occur after all the interactions of the CF LOOP.

[1] we adopt this notation to be compliant with the widely used inter-action formalism.

3 EXTENSION OF THE CAUSAL SEMANTICS

As indicated in the Sec. 2.1, the behaviour of the SD with the CF is deducted by flattening them and by applying weak sequencing. Accordingly, we obtain basic SD, and we face again the problem of the loss some behaviours and the inadequacy of standard semantics UML2.0 for the distributed systems. In our approach, the default composition operator, which is applied to CF, follows the causal sequencing. The partial order derived between the events is justified because we define the rules which govern the computation of this order. Moreover, with this partial order we obtain all possible behaviours for a given distributed system. The extension of causal semantics which we propose lies: *i)* in new rules of causal semantics for SD with CF and their formalization; *ii)* the definition of an event definition which belongs to a given SD with CF. The obtained traces are sequences of event occurrences. Each event is described by a definition and its occurrence depends on defined rules. In (Tahir et al., 2005), they propose rules of causal semantics only for basic diagram, and an event which belongs to a given SD can be executed if its preceding events are executed. We have extended the precedence relationship since for example, a given event which belongs to SD with an ALT or an OPT CF can have preceding events which can be executed only once or which can be ignored; a given event which belongs to SD with a LOOP CF can have preceding events which can be executed several times.

3.1 Formalization of the Causal Semantics

Our proposed specification supports SD containing sequential CF. In a SD with an ALT CF, we can have several CI evaluated to true at the same time, only one of them will be executed in a nondeterministic way. In the following, we give a formal definition of SD and different concepts used in this paper.

Well Formed Messages. We define M a set of messages, and EVT a set of events that is associate to messages. The set M is well formed if every message is identified by a pair of events: a sent event and a received event.

Conditional Combined Fragment (ALT, OPT). A conditional CF is a sequence of couples $F = < (guard_1, OP_1), (guard_2, OP_2), ..., (guard_j, OP_l) >$ where: $guard_i$ is the constraint which guards the i^{th} operand, OP_i is the set of events covered by the i^{th} operand.

Combined Fragment (LOOP). A CF LOOP is a quadruple $F = [guard, min, max, OP]$ where: $guard$ is the constraint which guards the loop operand, *min* is the minimum number of iteration, *max* is the maximum number of iteration and *OP* is the set of events covered by a LOOP CF. $ran(E)$ and $card(E)$ denote respectively the range and the cardinal of a set E.

Sequence Diagram. A sequence diagram is a tuple $SD : [O, M, EVT, FCT\_s, FCT\_r, FCT\_o, F, <_{Caus}]$ where:

- O, M, EVT are respectively a finite and non-empty set of lifelines, messages and events,
- E_s, E_r denote respectively the set of sent events and the set of received events,
- FCT_s, FCT_r are two bijective functions that associate respectively for each message a sent event and a received event,
- FCT_o: is total surjective function that associates for each event one lifeline representing the transmitter or the receiver,
- $F = < F_1, F_2, ..., F_k >$ is the sequence of k CF ALT, OPT, and LOOP,
- $<_{Caus}$: denotes the partial order relationship which is transitive, acyclic and non-reflexive.

We define the local order relation between events within each lifeline noted $<^*_{DS,o}$, such that $<^*_{DS,o}$ is the set of pair of events e, e', such that the events e, e' belong to the set EVT; they are directly consecutive, and are sent or received by the lifeline o. The local relation $<^*_{DS,o}$ is a minimal acyclic and non-transitive relation.

3.2 Case Study

We illustrate our approach through a case study depicted in Fig. 1. We model the interactions in a restaurant with UML2.0 sequence diagrams. The restaurant system is a distributed system since its components are independent: the client, the head_cook, the waiter1 and the waiter2. The client orders meal to waiter1 which transmits the order to the head_cook. Once the meal is ready, the head_cook alerts the waiter at most 3 times. The client orders a drink to waiter2 which will serve it. Then, the client asks for the bill for the waiter1 which in turn asks for the type of payment the client wants to make. The client has several alternatives: he can pay by cash, by card or by cheque. In the first case, if the paid amount is greater than the amount of the bill, the waiter1 gives change to the client. Once the bill was paid, the waiter delivers a receipt to the client.

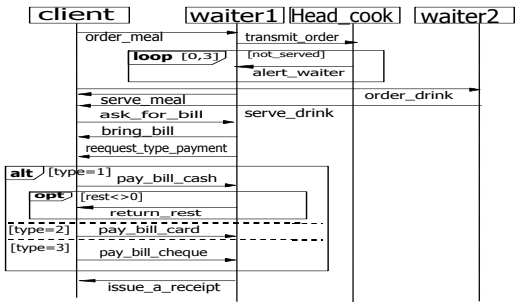

Figure 1: SD Restaurant.

3.3 The Partial Order Relationship $<_{Caus}$

The event occurrence depends on the partial order relationship $<_{Caus}$. We extend this relationship by redefining the causal rules to support CF.
We consider an example of SD with two CF $F =< F_1, F_2 >$ where: *i)* $F_1 = < (guard_1, OP_1), (guard_2, OP_2) >$ is an ALT CF which models only two possible behaviours however, the proposed solution is valid for an ALT CF with k operand. *ii)* $F_2 = < guard, 0, N, OP3 >$ is the LOOP CF. We consider the set $OP = OP_1 \cup OP_2 \cup OP_3$.
We define the total function F_{op}, that associate to each operand, in a given SD, the corresponding CF. We define the new relation $<_{guard}$, such that if the CI of CF depends on the execution of an event, which precedes CF then, we add a causality relation between this event and the first event authorized for execution in the ALT CF. Note that, the first event of the CF authorized to be execute is not necessarily the event which appears the first in the CF. For the rest of paper, we designate this event by the first event of the CF. In SD with sequential CF, the events may be located outside or inside the CF (ALT, OPT or LOOP). The events located outside the CF are of two categories: they can precede or succeed the CF. The events succeeding the CF may belong to lifelines covered or not by the CF. The proposed categorization of events will help us to define the causal rules for a SD with CF.

3.3.1 Computation of Causal Relationship

The causal semantics is defined as the union of these relations: $<_{Caus} = <_{sync} \cup <_{RE} \cup <_{EE} \cup <_{guard}$. For a given SD with sequential CF, the computation of causal relation is done in several steps. First of all, we apply the obvious rule which synchronizes the sending and receiving of each message ($<_{sync}$). Then, we omit all the CF of the SD, and we apply the rules of causal semantics; indeed as we have mentioned before, if the CI(the guard) is evaluated to False, then, none event inside the operand occurs. The case where all the guards of each CF are False is possible. According to the ALT CF, for every operand we omit the other operand of the CF, and we compute the relationships between the events which are inside the CF and other events. For the LOOP CF, there is no special treatment as for ALT CF. Following this approach, we solve the aberrant relations) and (deadlocks events) problems that we have already mentioned.

3.4 The Causal Rules

In this section, we give the conditions which must be satisfied by all pairs of events in both relations $<_{RE}$, $<_{EE}$. For each relation, we identify four rules.
$<_{RE} = \{(e,e' | rule1.1 \vee rule1.2 \vee rule1.3 \vee rule1.4\}$
$<_{EE} = \{(e,e' | rule2.1 \vee rule2.2 \vee rule2.3 \vee rule2.4\}$
Where in rule$i.j$, i denotes the relation (1 denotes the relation $<_{RE}$, 2 denotes the relation $<_{EE}$), j denotes the rule. We explicit these rules in the following. We identify four situations, for both relation, depending on the localisation of the events to be ordered. For all rules, the events e, e' belong to the same lifeline.

- The first situation is defined when the events e, e' to be ordered are located outside the CF.
  Rule 1.1: $<_{RE}$. If an event e" exists between the events e, e', it is necessarily either a received event or it belongs to a CF, located between e, e' (since the events of the CF can be ignored, in the case where the CI is False).
  $(\forall e)(\forall e')[(e,e') \in (EVT \setminus OP)^2 \wedge \exists\, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge e \in ran(FCT\_r) \wedge e' \in ran(FCT\_s) \wedge$
  $e <^*_{DS,o} e' \wedge (\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e') \Rightarrow$
  $e" \in ran(FCT\_r) \vee e" \in OP)]$.
  Rule 2.1: $<_{EE}$. If an event e" exists between e, e', it belongs necessarily to the CF located between e, e'.
  $(\forall e)(\forall e')[(e,e') \in (EVT \setminus OP)^2 \wedge \exists\, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge (e,e') \in (ran(FCT\_s))^2 \wedge$
  $e <^*_{DS,o} e' \wedge (\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e') \Rightarrow e" \in OP))]$
- The second situation is defined when the event e is inside to the operand OP_i, for which the CF is True, and the event e' succeeds the CF.
  Rule 1.2: $<_{RE}$. If an event e" exists between e, e', it is necessarily a received event or it belongs to one operand different from OP_i, if it is related to a CF ALT.
  $(\forall e)(\forall e')[e \in OP_i \wedge e' \in (EVT \setminus OP) \wedge \exists\, o \in O \wedge FCT\_o(e) =$
  $FCT\_o(e') = o \wedge e \in ran(FCT\_r) \wedge e' \in ran(FCT\_s) \wedge e <^*_{DS,o} e' \wedge$
  $\wedge (\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e') \Rightarrow e" \in ran(FCT\_r) \vee e" \in$
  $(OP \setminus OP_i))]$
  Rule 2.2: $<_{EE}$. If an event e" exists between e, e', it belongs necessarily to another operand different from OP_i of the considered CF.
  $(\forall e)(\forall e')[e \in OP_i \wedge e' \in (EVT \setminus OP) \wedge \exists\, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge (e,e') \in (ran(FCT\_s))^2 \wedge e <^*_{DS,o} e' \wedge$

$\forall(e")(e" \in EVT \ \wedge \ (e <^*_{DS,o} e" <^*_{DS,o} e') \ \Rightarrow \ e" \in (OP \setminus OP_i))]$
The third and the fourth situation concern SD with CF ALT or OPT.

- The third situation is defined when either e, e' are both in the same operand OP_i, for which the CF is True, or the event e precedes the CF, and the event e' is inside to OP_i.
  Rule 1.3: $<_{RE}$. If an event e" exists between the events e, e', it is necessarily a received event or it belongs to an operand different from OPi.
  $(\forall e)(\forall e')[e \in (EVT \setminus (OP \setminus OP_i)) \wedge e' \in OP_i \wedge \exists \, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge e \in ran(FCT\_r) \wedge$
  $e' \in ran(FCT\_s) \wedge e <^*_{DS,o} e' \wedge (\forall e")(e" \in EVT \wedge$
  $(e <^*_{DS,o} e" <^*_{DS,o} e') \Rightarrow e" \in ran(FCT\_r) \vee e" \in (OP \setminus OP_i))]$.
  Rule 2.3: $<_{EE}$. If an event e" exists between e, e', it belongs to an operand different from OPi.
  $(\forall e)(\forall e')[e' \in (EVT \setminus (OP \setminus OP_i)) \wedge e \in OP_i \wedge \exists \, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge (e,e') \in (ran(FCT\_s))^2 \wedge e <^*_{DS,o} e'$
  $\wedge (\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e) \Rightarrow e" \in (OP \setminus OP_i))]$.

- The fourth situation is where the events e, e' belong to operands OP_i, OP_j, for which the CFis True, of two sequential CF.

  Rule 1.4: $<_{RE}$. If an event e" exists between the events e, e', it is necessarily a received event or it belongs to other operands of CF different from OP_i and OP_j.
  $(\forall e)(\forall e')[e \in OP_j \wedge e' \in OP_i \wedge F_{op}(OP_j) \neq F_{op}(OP_i) \wedge \exists \, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge e \in ran(FCT\_r) \wedge e' \in ran(FCT\_s) \wedge$
  $e <^*_{DS,o} e' \wedge (\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e')$
  $\Rightarrow e" \in ran(FCT\_r) \vee e" \in (OP \setminus (OP_i \cup OP_j)))]$.
  Rule 2.4: $<_{EE}$. If an event e" exists between e, e', it necessarily belongs to the other operand of CF different from OP_i and OP_j.
  $(\forall e)(\forall e')[e \in OP_j \wedge e' \in OP_i \wedge F_{op}(OP_j) \neq F_{op}(OP_i) \wedge \exists \, o \in O \wedge$
  $FCT\_o(e) = FCT\_o(e') = o \wedge (e,e') \in (ran(FCT\_s))^2 \wedge e <^*_{DS,o} e' \wedge$
  $(\forall e")(e" \in EVT \wedge (e <^*_{DS,o} e" <^*_{DS,o} e') \Rightarrow e" \in (OP \setminus (OP_i \cup OP_j)))]$.

3.5 Behaviour of Sequence Diagrams

The behaviour of a given SD is a set of traces. The set of traces which defines a choice is the union of the guarded traces of the operands. The trace is a set of events occurrences. The occurrence of an event depends on its definition.

3.5.1 The State of Event

In the standard semantics and in the causal semantics, an event which belongs to a basic SD is either executed or not yet executed. Therefore, we define a function to assign to each event a state. The state of an event which is not yet executed is 1. After its execution the event is consumed, and its state becomes 0. For an event which belongs to a SD with a CF , we extend the state to allow us to deal with CF compactly. Each event which belongs to a SD with an ALT CF , can have one state among three: either 1 if it is not yet executed, or 0 if it is executed, or -1 if it is ignored. Initially, the state of the events which belongs to the LOOP CF is N, such that N denotes the maximal number of iteration. After the execution of each event of the LOOP CF, the state of each of them is decreased. If these events may be executed N times, or may not finish all the iterations, where the CI becomes False, their state must be 0, to allow the events which follow the CF to occur. Thus, an event must have one of the state values $0, 1, -1$ or N.

3.5.2 The Fictitious Event

The LOOP CF has exactly one operand, we choose to evaluate the CI only once in the first event which is authorized to occur; however, the ALT CF may contain several operands, we choose to evaluate the CI only once in the first event which is authorized to occur. Thus, if the CI is evaluated to True, and the causality constraints are satisfied then, this first event will occur and the events which depend on it may occur. Otherwise, if the CI is evaluated to False, and the causality constraints are satisfied, then the events which follow the CF must be executed, and the events of the CF will be ignored. Regarding the ALT CF, if it is ignored, then all its events must be disabled $(state = -1)$. Hence, the need to use the fictitious events, which will occur when the CI is evaluated to False. The effects of their execution consist in disabling the events of the CF. For the LOOP CF, if the CI is False, the fictitious event allows the other events inside the LOOP CF to finish their iteration, and to prevent them from executing the remaining iterations by decreasing their states.

3.5.3 The Definition of an Event

The behaviour of a SD, observed through, is the occurrence of its events. An event occurs under some conditions, and produces effects. Each event which belongs to a SD with CF has trigger conditions and execution effects. The trigger conditions include causality with other events (which must be formalized).

For an event, which does not belongs to none CF, the trigger conditions consist in *i)* the precedent events of the current event are executed or ignored, *ii)* the current event is not yet executed. The execution effect consist in updating the state of the event. For the other identified kind of events, in addi-

tion to these trigger conditions and the execution effects, we have other ones.

For the CF ALT, we distinguish two special kinds of events. The first one, is the first event of each operand of the CF, for which, we must add a trigger condition to evaluate the CF. And the execution effect, for modifying the state of the events, which belong to the second operand to -1, in order to ignore them (if it is about the first event of the first operand, otherwise we modify the state of the events which belong to the first operand to -1 in order to ignore them). The second one, is the fictitious event, for which, the second trigger condition consists in verifying that the CI of the CF is False. The execution effect consists only to modify the state of the events which belong to the CF to -1 to ignore them. For the CF LOOP, we distinguish three special kinds of events. The first one, is the first event of the CF, for which, we must add two trigger conditions, which consists in, evaluating the CI, and verifying that the current event was executed a number strictly greater than 0. The second kind are the other events inside the CF, for which, the second trigger condition is different and consists in verifying that the current event was executed a number strictly greater than 0. Moreover, we add the trigger condition which consists in verifying that the iteration number of the preceding events of the current event is lower than its number of iteration. The third kind is the fictitious event, for which, the second trigger condition consists in verifying that the iteration number of the preceding events of the current event is lower than its number of iteration. Moreover, we add the trigger condition which consists in verifying that the CI of the CF is False. The execution effect consists in updating the state of all events of the CF.

Up to now, we have extended the causal semantics, and defined accordingly the behaviour of SD with CF.

4 IMPLEMENTATION IN EVENT-B

In this section, we show how we proceed to implement in Event-B the SD which is equipped with extended causal semantics.

4.1 The Proposed Generic Event-B Specification

In an Event-B model we have two parts: a static part composed by the context and a dynamic part composed by the B-machine.

4.1.1 Construction of Contexts CTX and CTXP

We define two contexts. In the first context CTXP, we define the configuration which is appropriate to the considered SD. In the second context CTX, which *extends* the first context, we define the sets, the constants, the axioms and the theorems which allow to define the typing of constants or to express general properties of the SD. We detail them in the following. The contents of this context are unchanged for any SD. The constants OBJECTS, MESSAGES, EVT represent respectively the lifelines, messages and the events of SD. The sent events and the received events are represented respectively by the set EVT_SEND and EVT_REC. In the constant BEGIN, we specify the first event authorized for execution in the SD.

4.1.2 Construction of Event-B Machine for Basic Sequence Diagram

The derivation of the SD in Event-B is straightforward. Indeed, we have the following similarities between the two formalisms: (Triggers conditions, **Guards**) and (Executions Effect, **Substitution**). In Event-B, a guard has a label (like G1, G2...); in the same way, the substitutions have a label (like S1, S2...). The clause SEES allows the B-machine to have a visibility on the static declarations which are made in the context. B-machine is composed by:
1) **Variables and Invariants:** we define two variables with invariants: *i) state* a total bijective function that expresses the state of each event: $state \in EVT \rightarrow \mathbb{Z}$, and *ii) current_object* expresses the transmitter or the receiver of the current event: $current\_object \in OBJECTS$;
2) **the initialization:** initially all the events are not yet executed: $state := \text{EVT} * \{1\}$. The variable *current_object* is initialized to the lifeline of the event authorized to be executed;
3) **the events** of B-machine are the same events of the SD; therefore, the B specification generated and the SD have the same trace (equivalence of traces).

4.1.3 Event-B Specification for Sequence Diagram with Combined Fragment

We consider the example of the SD, depicted in Fig. 1, with two sequential combined fragments LOOP, ALT; the ALT CF is composed from three operands, and its first operand contains an OPT CF. However, the B specification which we propose is suitable for a SD with several sequential CF and notably where the ALT CF contains several operands. We keep the same structure as that proposed for basic SD, we outline what will change in the content of contexts and machine. We add in the contexts CTX and CTXP,

three constants EVT_OP1, EVT_OP2, EVT_OP3 representing respectively the events of the first, the second and the third operand of the ALT CF. We add a constant EVT_LOOP representing the events located inside the LOOP CF. The state of the event, which are inside the LOOP CF, must be initialized to N, and the events which are not inside the LOOP CF must be initialized to 1: $state := (\text{EVT} \setminus \text{EVT\_LOOP}) * \{1\} \cup \text{EVT\_LOOP} * \{N\}$

In the following, we give the implementation of the definition of an event, that we have already informally explained in the Sec. 3.5.3. An event is defined by its guards and substitutions.

As example of an event, which does not belong to a CF, we have *R_ordermeal*. The guards are: **G1.** $ran(Caus^{-1}[\{R\_ordermeal\}] \lhd state) \subseteq \{-1,0\}$ and **G2.** $state(R\_ordermeal) \geq 1$. The substitutions are: **S1.** $state(R\_ordermeal) := state(R\_ordermeal) - 1$ and **S2.** $current\_instance := FCT\_o(R\_ordermeal)$.

For the other identified kind of events, in addition to these obvious guards and substitutions cited above, we have the following ones. Regarding the CF ALT, as we have mentioned we distinguish two special kinds of events: for the first kind of event, which is the first event of each operand of the CF, example *E_paybillcash*, we add the guard to evaluate the CI. And the substitution: **S3.** $state := state \lhd\!\!- (((EVT\_ALT2 \cup EVT\_ALT3) * \{-1\}) \cup \{(E\_paybillcash \mapsto (state(E\_paybillcash) - 1))\})$. For the second kind of event, which is the fictitious event, example *E_paystop*, its guards are: **G1.** $ran(Caus^{-1}[E\_paystop] \setminus (EVT\_OP1 \cup EVT\_OP2 \cup EVT\_OP3) \lhd state) \subseteq \{-1,0\}$, and **G2.** $C = FALSE$. Its substitution is **S1.** $state := state \lhd\!\!- (EVT\_OP1 \cup EVT\_OP2 \cup EVT\_OP3) * \{-1\}$.

Concerning the CF LOOP, as we have mentioned we distinguish three special kinds of events: for the first kind of event, which is the first event of the CI, example *E_alertwaiter*, we add a guard to evaluate the CF **G3.** $taking\_into\_account = 0$; and a guard **G4.** $\forall(ee)(((ee \in EVT\_LOOP) \wedge (ee \in Caus^{-1}[\{E\_alertwaiter\}])) \Longrightarrow (state(ee) < state(E\_alertwaiter)))$. For the second kind of event, which is, the other events inside the CF, we add the guard **G3.** $(\forall ee) \cdot ((ee \in EVT \wedge Caus^{-1}[x]) \Rightarrow state(ee) < state(x))$. For the third kind of event, which is, the fictitious event, we must add two guards **G3.** $(\forall ee) \cdot ((ee \in EVT\_LOOP \wedge ee \in (Caus^{-1}[\{E\_alertwaiter\}])) \Longrightarrow (state(ee) < state\_r1(E\_alertwaiter)))$, **G4.** $taking\_into\_account = 1$. The substitution is **S1.** $state := state \lhd\!\!- \{x \mapsto y | x \in EVT\_LOOP \wedge y = state(E\_alertwaiter) - state(E\_alertwaiter))\}$.

The current results on the implementation of SD in Event-B allow us to start studying properties, and to check the relation of refinement between two SD, which model complex behaviours of a given distributed system.

5 TOWARDS THE REFINEMENT OF SEQUENCE DIAGRAMS

Event-B offers a powerful tool (Rodin), which can check the refinement relation between two given models, by generating proof obligations that must be proved. Consider two sequence diagrams SD1 and SD2 equipped with the proposed extended causal semantics; in order to verify that SD2 refines SD1 with Event-B: we translate independently each SD into a B-specification; we define the rules of refinement as well as the refinement properties which must be proved. The latter will be expressed as theorems. The proof of the refinement properties, and the validation of both obtained models allow to deduce, or not, that SD2 is the refinement of SD1.

Properties. The checking of properties is made conjointly by the tools Rodin and ProB, which are complementary. Rodin generates proof obligations (PO) that can be checked automatically or interactively. With Rodin, we checked especially static properties, that allow to prove the well formedness of the SD. These properties are expressed in the context of B-machine as theorems. For example, we verify that the SD is not empty: $Finite(OBJECTS)$, $Card(OBJECTS) \geqslant 2$, $(OBJECTS) \neq \varnothing$, $MESSAGES \neq \varnothing$, $EVT \neq \varnothing$; each event belongs to one lifeline: $Fct\_o \in \text{EVT} \twoheadrightarrow OBJECTS$; the causal relation is acyclic and non-reflexive $Caus \cap EVT \lhd id = \varnothing$. Dynamic properties are expressed in the clause invariant of B-machine. We check some safety properties such as all sent messages are received; each received event must be send before: $\forall ee \cdot (ee \in EVT\_REC \wedge state(ee) = 0 \Longrightarrow state(FCT\_EVT\_s(FCT\_r)^{-1}(ee)))) = 0))$. We have not deadlock (liveness property): at any moment we have an event authorized to be executed (this property is expressed by the disjunction of guards of all the events). The property of non-divergence of the events, which belong to LOOP CF, consist in ensuring that these events do not take infinitely the control. It is done by using the PO for refinement checking, and by defining an expression named *variant*, that must by decreased by the events running in the LOOP CF. In the given case study, the variant is: $state(E\_alertwaiter) + state(R\_alertwaiter)$.

We export the B-specification from Rodin to the ProB for the further checking. ProB supports automated checking of B-machines by model checking, and provides counter-examples when there is a violation of the invariant. In ProB, we can express fairness properties by means of linear temporal logic (LTL) formulas.

6 RELATED WORKS

Many researchers proposed semantics for sequence diagram for various purposes. In (Micskei and Waeselynck, 2011) a survey on semantics of SD to overcome problems that are not yet resolved by OMG. Other semantics (Aredo, 2000), (Cengarle et al., 2006), (Haugen et al., 2005), (Störrle, 2003) were proposed to verify some SD properties such as safety properties like deadlock freedom, or to verify the properties of the relation of refinement of sequence diagrams, like in (**?**), (Haugen et al., 2005), (Störrle, 2003). There are approaches which have defined semantics by translating the SD into other formalisms to benefit from their advantages. In (Whittle and Schumann, 2000) statecharts were generated from a set of SD and a set of OCL constraints. In (Cardoso and Sibertin-Blanc., 2001), (Eichner et al., 2005) SD were translated into Petri nets formalism. In (Grosu and Smolka, 2005) SD were translated into B*ü*chi automata. However, by making the translation some essential features of the SD are lost. These semantics are not suitable for modeling all types of systems especially for distributed systems. This mismatch has motivated the work of (Sibertin-Blanc et al., 2005), (Tahir et al., 2005) who proposed a causal semantics for basic SD. Many works (Laleau R., 2002), (Akram, 2006) have coupled UML and B-method, in two ways, either to ensure the traceability between the two formalisms or to check some properties and to eliminate inconsistency of UML diagrams ((Lano et al., 2001), (Ledang and Souquières, 2002)). The existing approaches provide semantics closely related to the target translation formalisms. In our approach, regardless of the translation language, we have considered an existing semantics for UML1.x SD, which is adequate for modeling of the behaviours of distributed systems; we have extended it to support UML2.0 SD with the most popular CF (ALT, OPT, LOOP). We take care in the translation to preserve all the essential aspects of the SD, by considering set theory and equivalence of traces shared by the formalisms.

7 CONCLUSION AND PERSPECTIVES

To help in preliminaries design steps of distributed systems, we have equipped UML2.0 sequence diagrams with an appropriate causal semantics. We have underlined the limitations of the existing UML2.0 semantics, which consists in *i)* imposing a total order on events in each lifeline, *ii)* flattening the combined fragments and, *iii)* the application of the weak sequencing to derive the traces of SD. Therefrom, we opted for the causal semantics initially proposed for basic SD in the setting of distributed systems. The causal semantics is based on partial order. We have proposed an extension of this causal semantics to cover the ALT, OPT and LOOP combined fragments. We overcome the insufficiencies of the standard UML2.0 semantics by proposing a treatment of CF which preserves their compact description: new behavioural rules are described and formalized. We have proposed an implementation in Event-B of the SD equipped with this extended causal semantics. The behaviour of SD (defined by their traces) can then be checked by analysing the Event-B models. The latter will also serve as the support for safety, liveness and fairness analysis. Several combinations of nested CF must be studied in order to generalize, and to extend our proposal to support nested CF; 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 invalid behaviours such as the NEG operator. Meanwhile, the current extension of causal semantics serves as the basis of our work on the verification of the refinement relation between sequence diagrams. The Rodin platform is extendable with plugins. We currently creating a plugin for checking the well-formedness of a given SD equipped with the extended causal semantics and for checking other properties. We are already conducting some experimentations on the refinement of SD.

REFERENCES

Akram, I. (2006). Couplage de spécifications B et de descriptions UML pour l'aide aux développements formels des systèmes d'information: Approche par méta-modélisation. In *Actes du 24ème congrés INFORSID*, Tunisie.

Aredo, D. B. (2000). Semantics of UML sequence diagram in PVS. Technical report, Online Proc. of UML2000 Workshop on Dynamic Behavior in UML models: Semantic Questions.

Cardoso, J. and Sibertin-Blanc., C. (2001). Ordering actions in sequence diagrams of UML. In *23rd Int. Conf. on Information Technology Interfaces*, pages 3–14.

Cengarle, M. V., Graubmann, P., Wagner, S., and Munchen, T. U. (2006). Semantics of UML 2.0 Interactions with Variabilities.

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*, 3530:133–148.

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.

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

Laleau R., P. F. (2002). Coming and Going from UML to B : A Proposal to Support Traceability in Rigorous IS Development. In *2nd International Conference of B and Z Users*, number 2272, pages 517–534. Springer.

Lano, K., Clark, D., and Androutsopoulos, K. (2001). UML to B: Formal verification of object-oriented models. In *Proc. 4th Intl. Conf. Integrated Formal Methods (IFM2004)*, number 2999, pages 187–206. Springer.

Ledang, H. and Souquières, J. (2002). Contributions for Modelling UML State-Charts in B. *Lecture Notes in Computer Science*, (2335):109127.

Micskei, Z. and Waeselynck, H. (2011). The many meanings of UML2.0 Sequence Diagrams: a survey. *Software & Systems Modeling*, 10(4):489–514.

Sibertin-Blanc, C., Tahir, O., and Cardoso, J. (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.

Störrle, H. (2003). Semantics of Interactions in UML 2.0. In *HCC*, pages 129–136.

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

Whittle, J. and Schumann, J. (2000). Generating statechart designs from scenarios. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 314–323. ACM Press.