Refinement of UML2.0 Sequence Diagrams for Distributed Systems

Fatma Dhaou¹, Ines Mouakher¹, Christian Attiogbé² and Khaled Bsaies¹

¹LIPAH, Faculty of Sciences of Tunis, Tunis, Tunisia
²LINA UMR 6241, University of Nantes, Nantes, France

Keywords: UML2.0 Sequence Diagrams, Refinement Relation, Sequence Diagrams Refinement, Event-B.

Abstract: Refinement process applied to UML2.0 Sequence Diagrams (SD) is adopted to deal with the complexity of modeling distributed systems. The various steps leading to the checking of the refinement of SDs theoretically as well as practically are explained. A refinement relation possessing the necessary properties, is formalized; its implementation in the Event-B method is proposed in order to check the correctness of the refinement of SDs, and to verify some safety, liveness properties and the termination of the new introduced events.

1 INTRODUCTION

Context. The UML2.0 sequence diagrams (SD) are widely used by designers thanks to their intuitiveness, the ease of graphical representation, and their high expressivity, allowing to model complex behaviour of systems. Refinement process is a privileged approach adopted by researchers to manage complexity. SDs must be refined correctly, this requires the formalization of refinement relation supporting an incremental development. Since the semi formality of SD didn’t allow such a verification, the coupling of a formal and tooled method supporting this process is recommended.

Motivation. The notion of refinement was well studied, in several works, for the labeled transition systems (LTS), and more generally for the modal transition systems (MTS) (Fischbein et al., 2006). However, few works are interested in the refinement of UML2.0 SDs, which are more and more used. Intuitively, refinement of SD consists either in substituting a lifeline by new lifelines and/or detailing the internal behaviour of a lifeline by adding new sub-lifelines and eventually new events. We outline two aspects for the SD: the structural aspect (lifelines, messages, events, CF) and the semantic aspect that is its traces. Mainly, the existing approaches that deal with refinement of SDs focus on one aspect and impose a restrictive hypothesis for the other aspect: i) most of them use a refinement relation which does not support explicitly the introduction of new events in the refined SD; these later are then hidden in order to verify the refinement relation, which can lead to some errors of modeling that might not be detected; ii) other approaches ignore or didn’t deal correctly with the guards of combined fragments (ALT, OPT, LOOP), in the formalization of the considered refinement relation that they consider; iii) most of the existing approaches did not propose efficient tools for the checking of refinement relation; iv) to our knowledge, all the approaches are based on trace semantics, and the verification of the refinement relation requires the computation of traces of SD pairs. This can lead to combinatorial explosion of number of traces to generate. Moreover, they are based on rules of standard semantics (Object Management Group, 2009) for the derivation of traces, that are not suitable for SD modeling distributed systems, since they take no account of the independence of components of distributed systems.

In our approach, SDs are equipped with an operational semantics based on causal semantics (Dhaou et al., 2015), which in turn are based on partial orders theory; this is suitable for distributed systems, permits to formalize the refinement relation straightforwardly and facilitates its verification.

The verification of the approach is absolutely necessary. The choice of a formal method that supports the refinement process (Schneider, Steve and Trehan, Helen and Wehrheim, Heike, 2012), which in turn are based on partial orders theory; this is suitable for distributed systems, permits the verification of the approach straightforwardly and facilitates its verification.
Contribution. The contribution of our work is manifold: we proposed an approach that deals with refinement of SDs, while preserving required behaviours and deals correctly with guards. We formalize a refinement relation favourable to an incremental development, it is based on existing ones that are already defined on guarded LTS. Finally, we propose a generic implementation with Event-B/Rodin-ProB, for checking of correctness of refinement relation.

Organization. The remainder of the article is structured as follows. In Section 2, we present a case study that is used to illustrate our approach. In Section 3 we first present an overview of the operational semantics of UML2.0 SD, then we formalize our refinement relation. Section 5 is devoted to the implementation of our approach in Event-B, we provided an overview on the methodology that allows us to verify the refinement between two SDs. Before concluding in Section 7, we present some related works in Section 6.

2 CASE STUDY

Through the following case study, we show a concrete example of an incremental development of behaviour of a distributed system. We model the interactions in a restaurant with UML2.0 SD (depicted in Fig. 1). The restaurant system has independent and distributed components: the Client, the Head_Waiter, the Kitchen and the Barkeeper. The Client orders meal and drink to the Head_Waiter. This latter transmits the orders respectively to the Kitchen and to the Barkeeper. Once the orders are ready, the kitchen and the Barkeeper solicit the Head_Waiter to serve it. Then, the Head_Waiter brings the bill. For bill payment, the Client has several alternatives: he can pay by cash, by card or by cheque. Once the bill was paid, the Head_Waiter delivers a receipt to the Client. A possible refinement is depicted in Fig. 2. The abstract lifeline Head_Waiter is substituted by the new lifelines Waiter1 and Waiter2. The events of Head_Waiter are distributed between Waiter1 and Waiter2. The lifeline Kitchen delegates the task of preparation of meal to the new lifeline Cook, which in turn, once the meal is ready, alerts the Kitchen at most 3 times. For simplicity, all the abstract events keep their same name in the refined SD.

3 SET THEORY NOTATIONS AND DEFINITIONS

This section introduces some necessary rudiments on UML2.0 SD as well as a part of the set theory notation that we use. The table 1 provides a summary of set theory notations. We consider a sub-set of SD containing sequential combined fragment CF.

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definition</th>
<th>Symbols</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &lt; )</td>
<td>Domain subtraction</td>
<td>( \rightarrow )</td>
<td>Partial injection</td>
</tr>
<tr>
<td>( \leq )</td>
<td>anti domain restriction</td>
<td>( \Rightarrow )</td>
<td>Total injection</td>
</tr>
<tr>
<td>( \subseteq )</td>
<td>range restriction</td>
<td>( \Rightarrow )</td>
<td>Partial surjection</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>total function</td>
<td>( \Rightarrow )</td>
<td>Total surjection</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>Bijective function</td>
<td>( \Rightarrow )</td>
<td>Relational overriding</td>
</tr>
<tr>
<td>( \Rightarrow )</td>
<td>Partial function</td>
<td>( \Rightarrow )</td>
<td>range of a set E</td>
</tr>
</tbody>
</table>

The formal definition of a SD is as follows:

**Definition 1** (Sequence Diagram). A sequence diagram is a tuple \( SD : (O,M,EVT,E,S,E,F,FCT,S,FCT,\sigma,F,C,F) \) where:

- \( O, M, EVT \) are respectively a finite and non-empty set of lifelines, messages and events. We consider
asynchronous messages, the set $M$ is well formed if every message is identified by a pair of events: a sent event and a received event.

- $E_{S}, E_{R}$ 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,
- $FCT_{0}$ is total surjective function that associates for each event one lifeline representing the transmitter or the receiver,
- $F = \langle F_1, F_2, ..., F_k \rangle$ is the sequence of $k$ combined fragments ($ALT, OPT, LOOP$),
- $\sim_{Caus}$ denotes the partial order relationship which is non-reflexive.

The definitions of $ALT, OPT, LOOP\ CF$ depend on guard, that is a predicate on variables, or can be a temporal constraint.

**Definition 2** (Conditional combined fragment ($ALT, OPT$)). A conditional combined fragment $F$ is a sequence of couples $F = \langle (guard_1, OP_1), (guard_2, OP_2), ..., (guard_k, OP_k) \rangle$ 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.

**Definition 3** (Combined fragment ($LOOP$)). A combined fragment $F$ is a quadruple $F = \langle guard, min, max, OP \rangle$ 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 $LOOP\ CF$.

### 4 REFINEMENT OF SEQUENCE DIAGRAM

In this section, we present the formalization of the refinement relation, and we enunciate the rules that govern a correct refinement between SD.

**4.1 Operational Semantics of Sequence Diagram**

Refinement based on trace semantics requires a meticulous work that consists in generating of all possible traces of SD, then in their categorization into required and possible traces (Lu and Kim., 2011) and (system Haugen et al., 2005); where possible traces correspond to all possible combinations of SD events, that are obtained by rules of derivation of traces of the considered semantics, and required traces are those that must be preserved when refining the SD. Moreover, these semantics are based on rules of standard semantics that are not suitable for SD that model distributed systems. The refinement of guard is not treated correctly. These drawbacks explain our orientation for defining refinement relation based on operational semantics that we defined on a previous work (Dhaou et al., 2015). (Dhaou et al., 2015).

The operational semantics that we proposed, is useful for the formalization of refinement relation. Indeed, an operational semantics is concretely given as a guarded transition system. Besides refinement relation is already well defined on transition system as a simulation relation. This advantage is exploited to express our refinement relation in Subsection 4.2. In addition, in our operational semantics, guards are supported intuitively. Our proposed operational semantics is different from existing approaches because it is based on the semantics for the scenario-based lecture named causal semantics; that is based on rules taking account of the independence of the objects involved to interaction, in a given distributed system. Intuitively, these rules are defined by the set of traces with some properties which can be summarized as follows: i) the events of the same lifeline are not necessarily ordered; ii) two successive event emitted by the same lifeline are ordered. In certain case of distributed systems, if they are addressed to the same lifeline, their receptions are ordered; iii) the reception of an event by a lifeline causes the emission of the event that is successive to it. These obvious rules was extended, in our previous work (Dhaou et al., 2015), to support SD with the most popular combined fragments ($ALT, OPT, LOOP$). Contrarily to the standard semantics, the computation of traces, in our work, for SD with combined fragment is made without flattening them.

According to this semantics, a trace is a sequence of event occurrences and it may be finite or infinite. We consider only the finite traces by excluding infinite loop. A trace depicts the history of message-exchange corresponding to executions of the system. Since, we deal with interleaving semantics, two enabled events to be executed may not occur at exactly the same time. We allow the same event to occur only once in the same trace, and $N$ times in the $LOOP\ CF$.

**Definition 4** (Trace). Let $EVT$ be the set of events in a sequence diagram SD. A trace of SD is a finite sequence: $<e_1, e_2, e_3, e_4, ..., e_n>$ where $e_i \in EVT$.

The operational semantics of a SD is the traces of events. It is given as a guarded transition system, that allows to take account of guard

$$\text{Sem}(SD) = \langle S, S^0, \triangle \rangle$$

where $S$ is the set of possible states of the SD, $S^0$ is the initial state, $\triangle$ the transition relation.
State. Each state of a SD is expressed with two variables (state, current instance) such that, the first one expresses the states of all events of SD, and the second one expresses the lifeline of the current event. The variable state is a function

\[ \text{State : EVT} \rightarrow \{-1, 0, 1, N\} \]

where 1 and N express that the event is not yet executed, and must be executed respectively 1 or N times; 0 and −1 express that the event is respectively consumed or ignored. Initially all the events are not yet executed (1 or N). The initial state \( S^0 \) of SD is when any event is not yet executed (1 or N). The final state of SD is when all events are occurred or ignored.

Transition Relation. The events of the guarded transition system correspond to the events of the considered SD and some fictitious events. Consequently, the transition system has two rules such that the first rule deals with the execution of a single event; and the second rule handles the fictitious events. The introduction of the fictitious events is necessary to allow a high flexibility in defining execution strategies for ALT, OPT and LOOP CF. We define the following set of fictitious events:

\[ \text{evt}_{fict} = \{ f_{\text{call}}, f_{\text{opt}}, f_{\text{loop}} \} \]

By disregarding fictitious events, the set of traces of SD is the same as the set of traces of its correspondent operational semantics. For each category of event we define a rule for the guarded transition system. For instance, for an event \( e \) in SD, we associate the following transition:

\[ p \xrightarrow{\text{Guard}_e} q \text{ def } (\exists (g, e, q) \in \Delta \land g) \]

An event \( e \) is enabled in state \( p \) only when its trigger conditions hold. The triggers conditions of an event \( e \) consist in: verifying the execution of its preceding events, it is not yet executed and its guard is true. After its execution, its state is decreased and the current lifeline is updated.

4.2 Formalization of the SD Refinement Relation

The existing approaches focus on one aspect of refinement by imposing restrictive hypothesis for the other aspect, this can be constraining. In our work, we tried to benefit of the advantages of the existing approaches for proposing an approach that takes account of both aspect of refinement (structural and semantic) by relaxing some hypothesis.

We consider SD with distinct alphabets. We define a refinement relation with respect to an explicit mapping between the abstract and refined labels (lifelines, events). It permits the introduction of new events and lifelines, and guarantees the preservation of required behaviours. In addition, the refinement relation supports intuitively guard since it is defined as a simulation between guarded LTS (Leuschel, Michael and Butler, Michael, 2005).

The verification of the correctness of the refinement requires the satisfaction of the rules of the refinement that must be defined beforehand, and formalized in the refinement relation. In the following, we explain the refinement of SD that we consider, and we give the rules of a correct refinement that are marked with the symbol (Ri). The rules are dependents. Structural refinement concerns the refinement of labels (lifelines, messages) of the abstract SD.

R1. The events of substituted lifelines must be inherent by the new substituted lifelines.

R2. The lifelines which are not refined have to keep the same events.

R3. New events, if they exist, must be exchanged only between new lifelines.

Indeed, each lifeline can be substituted by new lifelines or its internal behaviour can be detailed by adding new sub-lifelines (child lifelines). For the first case of lifeline refinement, it is obvious that the abstract events must be distributed between new lifelines (R1), and the not-refined lifelines must keep its abstract events (R2). Evidently, the new lifelines can exchange between them new events (R3).

R4. By remaining in the same level, a behaviour of a given lifeline can be detailed. Its sub-lifelines must exchanged exclusively messages with it.

In the second case of lifeline refinement, the internal behaviour of the refined lifeline (parent lifeline) is not visible by the other lifelines. Hence, the child lifelines must exchange exclusively new events with their parent lifeline (R4), and the parent lifeline keep the same abstract events (R2).

R5. The new events must terminate.

In order to make verification of refinement relation. The new events are considered as silent events (\( \tau \) events).

Our approach supports semantic refinement by the reduction of abstract possible traces in the refined SD. This is achieved by the following methods:

i) changing the order of appearance of some abstract messages in the refined SD. This is allowed by our semantics since the events are not ordered along lifelines. For example, in the refined SD of Fig.2, the message order_drink is moved after the message serve_meal.

ii) reduction of non-determinism for guarded CF: this is done by strengthening the guards with supplementary conditions (C). For instance, in the
refined SD of Fig.2, the guards of the second and the third operand of ALT CF are strengthened such that 
guard2 becomes (type = 2 ∧ m ≥ 20) and guard3 becomes (type = 3 ∧ m ≥ 20); where m denotes 
the amount_bill. iii) adding some CF like STRICT or ASSERT to cover some abstract events in the refined 
SD;

R6. All the events of SD1 must be present in SD2 up to a renaming.
To preserve branches of ALT CF, the rule R6 must be applied. Indeed, by applying one method among them 
we must ensure that each refined trace must have its counterpart in the abstract SD, by disregarding new 
events. Some approaches, proposed the suppression of some branches of ALT CF (Kruger, 2000) in 
order to eliminate non-determinism. However, the non-determinism offered by branches of alt reflects 
choices and must be kept during refinement. But it can be reduced by strengthening guards.

We consider two sequence diagrams SD1 and SD2 such that: SDi = ⟨O, Mi, EVT, Ei, Ai, ECI, FCTi2, FCTi1, Fi, SemSDi⟩, 
and their operational semantics8Sem(SDi) = ⟨δi, Si, Ai, Causi⟩ with 
i ∈ {1, 2}.

Structural Refinement. The preliminary step of 
the refinement checking is the definition of the mapping relation between the considered SDs; this 
requires the software designer’s intervention. A mapping 
relation between SD1 and SD2 is defined as a 
mapping between their lifelines and their messages. 
A lifeline O can be either substituted by lifelines, or 
its internal behaviour can be detailed by adding new 
sub-lifeline(s). Hence, a mapping relation must be 
defined between the abstract lifeline and refined lifelines. Each abstract lifeline must be linked to one or 
more lifelines, that can be either not refined lifelines or new lifelines (substituent or child lifelines).

Definition 5 (Lifeline mapping).
We define the function, lifeline mapping χ, that is a 
partial surjective function: χ : O2 ↦ O1.

All the abstract messages must be present in the refined 
SD, and the refined SD may contain new messages.

Definition 6 (Message mapping).
We define the function ρ : M2 ↦ M1, which is a 
partial bijection, such that its domain is the set of the 
refined messages, and its codomain is the set of the 
abstract messages.

For simplicity, we overload the partial bijection ρ as 
supports to follow events.

We introduce a new set New_M2, and New_EVT2 
such that New_M2 = M2 \ ρ⁻¹(M1) and New_EVT2 = 
EVT2 \ ρ⁻¹(EVT1) to identify respectively the new 
introduced messages and new events in refined diagram.

Proposition 1. (Does SD2 ⊑ SD1?)
Let SD1 and SD2 be two sequence diagrams in SD. 
We define two functions χ and ρ. SD2 is a structural 
refinement of SD1 with respect to χ and ρ, denoted 
SD2 ⊑ SD1, if the following conditions hold:

• the abstract messages in the SD2 are mapped to 
its correspondents in the SD1 by means of the 
function χ; M1 ⊆ ρ(M2);

• the abstract lifelines must keep all their abstract 
events, and the substitute lifelines inherit the ab-
stract events in SD2:
FCTi1 = ρ⁻¹(EVT1) ◦ FCTi2; χ

• the new messages exchanged between new life-
lines in the SD2 are linked to the same abstract 
line:
(New_M2) ◦ FCTi2; FCTi2; χ = 
(New_M2) ◦ FCTi2; FCTi2; χ

Each item of Prop.1 refers to one or more rules de-
defined above. The first one references R6; the second 
one deals with R2, R1 and the third one refers to R3. 
This relation is in accordance with Event-B refine-
ment relation. Indeed, we support a refinement re-
lation based on alphabet translation, that requires an 
explicit mapping between labels of SD pairs. New 
events in refined SD are considered as silent and re-
placed by τ transitions. We introduce the mapping re-
lation between operational semantics pairs associated 
to SD pairs.

Semantic Refinement. The refinement relation is 
defined as a simulation on operational semantics. Two 
models may be compared only if they have the same 
alphabet. Therefore, the new events are considered as 
silent and replaced by τ transitions.

Definition 7 (States mapping).
Let Sem(SD1) and Sem(SD2) be two operational se-
manics associated to SD pairs. The mapping relation 
R between pairs operational semantics consists in 
linking their respective states Si with respect to both 
functions χ and ρ. Si depends on values of the 
two variables current_instance.i and state.j; where 
i ∈ {1, 2}
each refined lifeline must be mapped with an abstract one:
current instance 1 = χ(current instance 2)

- by disregarding new introduced events, each abstract event, in the refined SD, must be mapped to its correspondent in the abstract SD:

\[ (\text{New}_{\text{EVT}}_{2}) \triangleright \text{state}_{2} = p, \text{state}. \]

**Proposition 2.** (Does SD2 ⊑ SD1?)

Let SD1 and SD2 be two sequence diagrams such that SD2 is a structural refinement of SD1 (SD2 ⊑ SD1), and Sem(SD1) = (S, S′, Δ) their operational semantics. Moreover, the new events of SD2 must terminate. SD2 is refinement of SD1 with respect to a map relation R(p, χ) denoted SD2 ⊑ SD1 if the following conditions hold:

- Initial states are linked : \( (s^0_1, s^0_2) \in R \)
- For each execution of an event e in SD2 which refines an abstract event \( e \in p^{-1}(M_1) \), there exists an execution of \( p(e) \) in SD1 that is relied to it, with strengthening of guard, in case of an event whose its execution depends on guard:

\[ (\forall e \in p^{-1}(M_1)) (\forall p \in S_1) (\forall q \in S_2) (\forall q' \in S_2) \]

\[ [(p, q) \in R \land q \xrightarrow{e} q' \Rightarrow (\exists p' \in S_1)[p \xrightarrow{e} p'] \land g \Rightarrow g' \land (q', p') \in R]] \]

- New events didn’t change the state of abstract SD.

\[ (\forall e \in \text{New}_{\text{EVT}}_{2}) (\forall p \in S_1) (\forall q \in S_2) (\forall q' \in S_2) \]

\[ [(p, q) \in R \land q \xrightarrow{e} q' \Rightarrow (q', p) \in R \]}

**Properties Preserved by Refinement Relation.** As consequence of the Prop.2, we have a trace refinement (Leuschel, Michael and Butler, Michael, 2005), by disregarding new introduced events in SD2, where the required behaviours are preserved. The refinement relation allows the extension of traces, by distinguishing between the observable and the non-observable events.

It is reflexive, since we have SD1 ⊑ SD1 that always holds. It is transitive, since SD2 ⊑ SD1 and SD3 ⊑ SD2 implies SD3 ⊑ SD1. Finally it is substitutive because it supports the renaming of abstract labels in the refined SD.

5 IMPLEMENTATION OF REFINEMENT RELATION IN EVENT-B

Our semantics is encoded in Event B (RODIN, 2007), in such way that proving the consistency (PO) of B models establishes the correctness of the refinement. For correctness analysis purpose, SD pairs to be checked are translated into Event-B specifications; these specifications are updated with the defined rules and analyzed with theorem prover Rodin, as well as checking model ProB. The choice of Event-B method is justified thanks to several advantages offered by it. Firstly, the existing similarities between both formalisms (Dhaou et al., 2015) provide an immediate translation of both SD formal definition, that is based on set theory and predicate logic, and its operational semantics, into Event-B specification. Secondly, Event-B refinement relation supports naturally the introduction of new events as well as the verification of their termination. Moreover, contrarily to the most refinement relations that require an implicit mapping between labels of SD pairs, Event-B refinement relation is based on alphabet translation which requires an explicit mapping between labels of SD pairs. Hence, errors of modeling are easily detected when the invariant is broken. Finally, Event-B has a powerful tool that allows the verification of the correctness of refinement SD. An Event-B specification is composed by a B-machine and a context. Event-B machine is mainly made of four elements: a name, a list of named predicates, the invariants, and the events. The context referenced by a B-machine is made of: a name, a list of distinct carrier sets, a list of constants and a list of named properties.

5.1 Translation of SD into Event-B Specification

Let SD : \( (O, M, EVT, E_{\text{Id}}, E_{\text{FCT}}, F_{\text{Id}}, F_{\text{FCT}}, F, C, \text{Caus}) \) and its operational semantics Sem(SD) = (S, S′, Δ). Firstly, we translate SD formal definition into contexts; to alleviate B specification and facilitate its proofs, we define two contexts such that: \( i \): in the first context CTX_XP, the concrete values the SD configuration (lifelines, messages, events) are expressed by sets and constants; \( ii \) in the second context CTX_X that EXTENDS the first one, properties that express that the considered SD is well-formed are marked as theorems. Secondly, the operational semantics of SD, Sem(SD), is translated into Event-B machine such that: \( i \): the State is translated into two variables: state, which is a total bijective function that expresses the state of each event, and current_instance, which expresses the transmitter or the receiver of the current event; \( ii \) each transition that belongs to Δ is represented by an event of Event-B machine.

5.2 Checking of the Correctness of Refinement between Two SD

Once we translate independently pairs SD into B-spe
specifications, we express the mapping relations, defined in Subsection 4.2 that should be proved statically, for some of them, and dynamically for the other ones. We add two new contexts $CTX_2P$ and $CTX_2$, such that in $CTX_2P$ we express the concrete mapping between pairs $SD$; and in $CTX_2$ we express the rules of structural refinement, in Subsection 4.2, that are marked as theorems to be proved. Tab. 3 depicts the expression of some structural refinement rules. For instance, $R3$ is expressed by axiom8. $R2$ and $R1$ are expressed by axiom9 and the lifeline mapping is expressed by axiom10. The refined machine is updated by adding the gluing invariant Tab.2 that expresses the state mapping.

Table 2: Gluing invariant.

| inv1: current_instance = FCT_INSTANCES(current_instance) |
| inv2: New_EVT $\triangleright$ state $\triangleright$ state |

Table 3: Some theorems of CTX2.

| axiom8: run(New_EVT $\triangleright$ R ; FCT $\triangleright$ New_INSTANCES |
| axiom9: A ; FCT $\triangleright$ ((A_EVT $\triangleright$ R ; FCT $\triangleright$ FCT_INSTANCES |
| axiom10: FCT_INSTANCES $\in$ R INSTANCES $\rightarrow$ A INSTANCES |

Table 4: The event $E_{order}$.drink.

```
E_order_drink: not extended extraordinary

r: EEE ((EE $\triangleright$ EVT2 $\triangleright$ EE $\triangleright$ Can2 $\sim$ E_order_drink)) $\Rightarrow$

W: state2(EE) $\triangleright$ state2(R_alart)) $\Rightarrow$

END
```

Concerning the events of the refined machine, we refund all the abstract events, which corresponds to message mapping, in addition to the new introduced events. In Tab. 4, we present an event of the refined machine: $E_{order}$.drink).

Verification of the Termination of the New Introduced Events. The new events which are introduced in the refined model must not take indefinitely the control. Event-B method allows to verify this property. This is done by the definition of an expression called variant. By demonstrating that the execution of each new event decreases the variant that must never goes below zero. Since a variant cannot be decreased indefinitely, this allows us to prove that a new event cannot take control forever. As example of new event we have $E_{alert}$. Its status is set to convergent, which means that for each execution of this event, the variant is decreased. In the case study, we have new events, on the one hand $E_{alert}$, $R_{alert}$ which can be repeated at maximum 3 times, on the other hand we have $E_{delegate}$ and $R_{delegate}$. We define for them the following variant: $(\langle New_{event} \triangleright state \rangle) \triangleright \{1\} \wedge N$. By proving the termination of new events, the rule $R5$ Sec. 4 is satisfied. When all proof obligations of B-specification are proved, we have a correct structural and semantics refinement.

The verification of refinement relation of $SD$ pairs is based on systematic rules of translation. The case study is totally implemented in Event-B on the basis of the generic architecture Fig. 3. The refinement is proved correct; through POs that were either automatically proved for some of them, or interactively proved for others ones.

6 RELATED WORKS

Mainly, existing approaches for building behavioural models of distributed systems by refinement are of two categories: the first one is based on formal frameworks, like Input/Output Automata (Peter M. Musial, 2012), the second one is based on scenarios like MSC, MTS, SD (Kruger, 2000). Despite the rigour of the approaches of the first category, they remain very hard which repress their use by engineers. The approaches of the second category are attractive and intuitive. Our approach belongs to this category of works.

Two important aspects must be considered for refinement $SD$: i) structural aspect that includes the different features of the SD (lifelines, messages, events, CF...); ii) semantic aspect that concerns traces. The existing approaches focus on one aspect by imposing strict hypothesis for the other aspects.

In (Ohlhoff, 2006), the authors treated the refinement of SD by defining strict rules of a correct structural refinement of SD that are impractical for de-
7 CONCLUSION

This work deals with refinement of SDs that, especially, model behaviors of distributed systems. To overcome the limitations of existing approaches, our proposed approach has the following advantages:

- it considers both structural and semantic refinements of SD: i) by defining clearly rules that govern a correct structural refinement guiding the designer; ii) by proposing an intuitive and coherent method for the formalization of refinement relation. This latter possesses the necessary properties favorable to an incremental development like transitivity; it permits the addition of new labels in the refined SD. It is substitutive since it supports the renaming of labels; which is a desired property in case of reuse of SD and their instantiation,

- it permits to reduce the set of possible abstract traces in the refined SD, while the required ones, expressed by ALT CF, are preserved,

- we deal correctly with the guards (of ALT, OPT and LOOP CF) as well as their refinement,

- our approach is not linked with a target formalism chosen for the implementation. The implementation processing in Event-B for correction of refinement relation is generic.

We are extending our proposal to consider SD with some CF that are particularly important for distributed systems like parallel operator and co-regions, as well as time features and nested CF.

REFERENCES


