Universitetet i Oslo
Institutt for informatikk

Time Exceptions in Sequence Diagrams

Oddleif Halvorsen, Ragnhild Kobro Runde, Øystein Haugen

Research Report 344
ISBN 82-7368-300-1
ISSN 0806-3036

Revised March 2008

Time Exceptions in Sequence Diagrams

Oddleif Halvorsen[1], Ragnhild Kobro Runde[2], Øystein Haugen[2]

[1] Software Innovation
[2] Department of Informatics, University of Oslo
{oddleif|ragnhilk|oysteinh}@ifi.uio.no

Abstract. UML sequence diagrams partially describe a system. We show how the description may be augmented with exceptions triggered by the violation of timing constraints and compare our approach to those of the UML 2.1 simple time model, the UML Testing Profile and the UML profile for Schedulability, Performance and Time. We give a formal definition of time exceptions in sequence diagrams and show that the concepts are compositional. An ATM example is used to explain and motivate the concepts.

Keywords: specification, time constraints, exception handling, formal semantics, refinement.

1 Introduction

UML sequence diagrams [9] are a useful vehicle for specifying communication between different parts of the system. A sequence diagram specifies a set of positive traces and a set of negative traces. A trace is a sequence of events, representing a system run. The positive traces represent legal behaviors that the system may exhibit, while the negative traces represent illegal behaviors that the system should not exhibit.

Timing information may be included in the diagram as constraints. These constraints may refer to either absolute time points (e.g. the timing of single events) or durations (e.g. the time between two events). The described behavior is negative if one or more time constraints are violated.

In practice, it may often be impossible to ensure that a time constraint is never violated, for instance when the constrained behavior involves communication with the environment. Usually, a sequence diagram does not describe what should happen in these exceptional cases. In this paper we demonstrate how the specification may be made more complete by augmenting the sequence diagram with exceptions that handle the violation of time constraints. The ideas behind our approach originate from [2], which treats exceptions triggered by wrong or missing data values in the messages.

Time violations are exceptional situations that are not supposed to happen very often. Modeling violation of time constraints as exceptions rather than using the alt-operator for specifying alternative behaviors, has the advantage that

– specifying the exceptional behavior separately from ordinary/expected behavior makes the diagrams simpler and more readable,
– exceptional behavior can easily be added to normal behavior in separate exception diagrams.

A single event may violate a time constraint by occurring too early, too late or never at all. All three situations will result in an exception, but the exact exception handling to be performed will typically be very different depending on the nature of the violation. Here we focus on the last case, where an event has not occurred within the given time limit and we therefore assume that it will not occur at all. If the event for some reason occurs at some later point it should be treated as another exception.

2 Background

In this section we motivate our work by presenting state of the art regarding timing constraints in UML. The main conclusion may be summarized as follows: Both the UML 2.1 simple time model (Sect. 2.1) and the UML profile for Schedulability, Performance and Time (Sect. 2.2) introduce concepts and notations for defining time constraints, but do not consider what should happen in case of violations. TimedSTAIRS (Sect. 2.3) distinguishes between the reception and the consumption of a message, but being based on UML 2.1 simple time model, TimedSTAIRS does not consider violations either. The default concept of UML Testing Profile (Sect. 2.4) and our previous work on exception handling (Sect. 2.5) consider violation of constraints, but mainly regarding wrong or missing data values, and not time constraint violations.

2.1 The UML 2.1 Simple Time Model

UML 2.1 [9] includes a simple time model intended to define a number of concepts relating to timing constraints. In general the semantics of the timing constraints follow the general interpretation of constraints in UML: "A Constraint represents additional semantic information attached to the constrained elements. A constraint is an assertion that indicates a restriction that must be satisfied by a correct design of the system." Furthermore the timing constraints always refer to a range of values, an interval. "All traces where the constraints are violated are negative traces (i.e., if they occur in practice the system has failed)." Some notation is introduced to define such interval time constraints and we apply this notation in this paper. UML 2.1 only states that when the constraints are violated the system is in error. Exceptions triggered by time constraint violations are not considered.

2.2 UML Profile for Schedulability, Performance and Time

The UML profile for Schedulability, Performance and Time Specification [7] is a profile based on UML 1.4 [6] describing in great detail concepts relating to timely

matters. The profile, hereafter referred to as SPT, will have to be updated to UML 2.1. There is now ongoing work to upgrade the real time profile under the name MARTE.

SPT introduces a large number of concepts. They represent most often properties of behavioral units needed for their scheduling and for performance analysis. Exceptions are not mentioned at all. By introducing concepts that allow to define "timing marks", it is possible to describe constraints on these timing marks, and in principle express time and duration constraints similar to what is the case with UML 2.1 simple time model. SPT allows constraints to be expressed on a large number of properties having been declared on behavioral units, but it never considers what happens if the constraint is not met. Implicitly this means that if the constraint is not met, the system is in complete failure.

2.3 TimedSTAIRS

TimedSTAIRS [4] is an approach to the compositional development of timed sequence diagrams. With time constraints, we argue that it is important to know whether a given constraint applies to the reception or the consumption of the message. Hence, in [4] we introduce a three-event semantics for timed sequence diagrams. In some cases, the time constraint should apply to the receiving of the message, while it in other situations should apply to the consumption.

In order to make a graphical distinction between reception and consumption, [4] uses a double arrow for reception and the standard single arrow for consumption. We will follow this convention in our examples. If only the consumption event is present in the diagram, the reception event is taken implicitly, while if only the reception event is present, the implicit consumption event may or may not take place.

In TimedSTAIRS, the semantics of a sequence diagram is a set of positive (i.e. legal) behaviors and a set of negative (i.e. illegal) behaviors. All traces that are not described in the diagram are said to be inconclusive. These may later supplement either the positive or the negative traces to refine the specification. Please see Sect. 4 for a more precise semantics.

2.4 UML Testing Profile — Default Concept

The U2TP (UML Testing Profile) [8] introduces the notion of Defaults that aims to define additional behavior when a constraint is broken. The notion of Defaults come from TTCN (Testing and Test Control Notation) [1] where it is used in a more imperative sense than sequence diagrams. In the UML Testing Profile the semantics is given by an elaborate transformation algorithm that in principle produces the traces of the main description combined with the Defaults on several levels.

However, U2TP says little about the semantics of defaults triggered by the violation of time constraints. The idea behind the defaults on different levels is that even the notoriously partial interactions are made complete and actually describing all behaviors. But the U2TP definition is not adequately precise in

this matter and there are no convincing examples given to explain what happens when a time constraint is violated.

2.5 Proposed Notation for Exceptions in Sequence Diagrams

In [2] we introduce notation for exceptions in sequence diagrams. The constraints that are violated are always on data values at the event associated with the exception. Violation of time constraints is not considered. The semantics of the behavior including the exceptions are given by a transformation procedure quite similar to that of U2TP. The idea is that supplementing traces are defined in the exception starting from the prefix of traces leading up to a triggering event.

The other novelty of our approach in [2] is that it suggests a scheme of dynamic gate matching that makes it possible to define exceptions independently. That idea is orthogonal to what we try to convey in this paper.

3 Time Exceptions in the ATM Example

An example with an Automatic Teller Machine (ATM) shows how time exceptions supplement the description and make the specification more complete and comprehensive without losing sight of the normal scenarios. The ATM example is based on the case from [2].

3.1 The Normal Flow

The normal flow refers to a happy day scenario when everything goes right. We show the use of an ATM to withdraw money. The user communicates with an ATM, which in turn communicates with the Bank.

Withdrawal in Fig. 1 specifies that the user is expected to insert a card and enter a four digit pin, whereas the ATM is to send the pin to the bank for validation. While the bank is validating the pin, the ATM asks the user for the amount to withdraw. If a valid pin is given, the bank will return OK. Then the ATM orders the Bank to withdraw the money from the account and gives the cash and the card to the user.

EnterPin in Fig. 1 specifies how the user gives the ATM the four digit pin. The loop(n) construct may be viewed as a syntactical shorthand for duplicating the contents of the loop n times. An interaction use (here: referring EnterPin) means the same as an inclusion of a fragment equal to the referred sequence diagram.

This specification is not very robust, and cannot serve as a sufficient specification for implementation. What if the user enters a wrong pin, the ATM is out of money, the user's account is empty or the ATM loses contact with the bank? We argue for the need to handle exceptions, even though sequence diagrams will always be partial description that are not supposed to cover every possible trace. Still, we aim at making the diagrams more complete, focusing on the important functionality of the system. Another goal is to make a clearer separation of normal and exceptional behavior and thus increase readability.

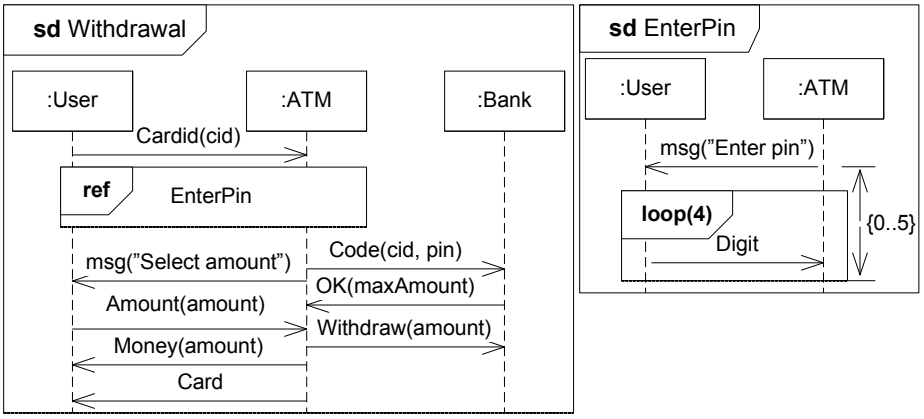

Fig. 1. Specification of withdrawal and entering a pin

3.2 Applying Time Exceptions to the ATM

Sequence diagrams are often filled with various constraints, but they seldom say much about what to do if a constraint breaks. Hence the system has completely failed if a constraint is broken. This is less expressive than desired. In order to make the specification more robust, we will add time exceptions to the ATM case. A time exception may be that the user for some reason leaves before completing the transaction, or that the bank spends too long time to validate the given pin.

As mentioned, time violations are of three kinds, either the event arrives too early, too late or never. Here we assume that if an event has not occurred within the specified constraint, it will never happen. If the event for some reason occurs after the constraint was violated it should be treated as another exception.

The semantics of time constraints builds on timestamps. We assume that the running system performs some kind of surveillance of the system, to evaluate the constraints. Intuitively, this means that we consider time constraints conceptually to behave like alarm clocks. If the associated event is too late the alarm goes off and the exception handler is triggered.

3.3 Time Exceptions in EnterPin

We present the notation by applying a time exception to the EnterPin diagram. An exception occurs when the user enters less than four digits or that the digits for some reason is not received by the ATM. If we do not handle this, the ATM will not be ready when the next user arrives. We need a way to decide whether the user has left, and then take the card from the card reader and store it some place safe before canceling the user's session.

In EnterPin in Fig. 2 we have added a time constraint stating that if the ATM has not received all the digits within the specified time, the exception UserLeftCard will fire. The time constraint itself is initialized on the send event on msg, and attached to the bottom of the loop fragment. Attaching it to the

bottom of the loop fragment indicates that the time constraint must hold for the last event, and hence all the preceding ones as well.

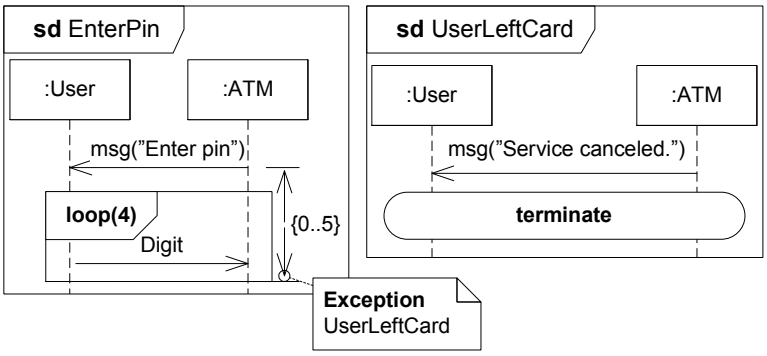

Fig. 2. EnterPin with time exception

UserLeftCard in Fig. 2 shows how the UserLeftCard exception is handled. In the case that the user leaves the ATM before proper completion of the service, the ATM sends a message stating that the service was canceled. By stating terminate we mean that the service, withdrawal of money, is to terminate — not the whole ATM. This will be explained in more detail below.

3.4 Time Exceptions in Withdrawal

In Fig. 3 we apply time exceptions to a more complex example to highlight some challenging situations.

Notice that the ATMPinValidation exception uses three-event semantics as described in TimedSTAIRS (see Sect. 2.3). This states that the message only needs to be received in the message buffer within the specified time constraint and not consumed. The reason for this time constraint is mainly to make sure that we do not lose contact with the bank during the request.

Fig. 4 specifies how an ATMPinValidationTimeout exception is handled by the ATM and the Bank. The exception is triggered if the ATM does not receive the result of the pin validation within the specified time. Our first exceptional reaction is to repeat the request to the Bank. If the response from the bank again fails to appear within the given time, the ATMCancel exception is triggered.

Fig. 4 illustrates that an exception may end with return or with terminate. While return means a perfect recovery back to the original flow of events, terminate means that the service should be terminated gracefully. Termination concludes the closest invoker declaring catch as shown in Fig. 3. If neither return nor terminate is given, return is assumed. If no catch is found, the system will not continue.

The events of a sequence diagram may in relation to an exception trigger be divided in three groups. First there are the events that have occurred before the trigger. Second we have events that must occur after the exception, and third

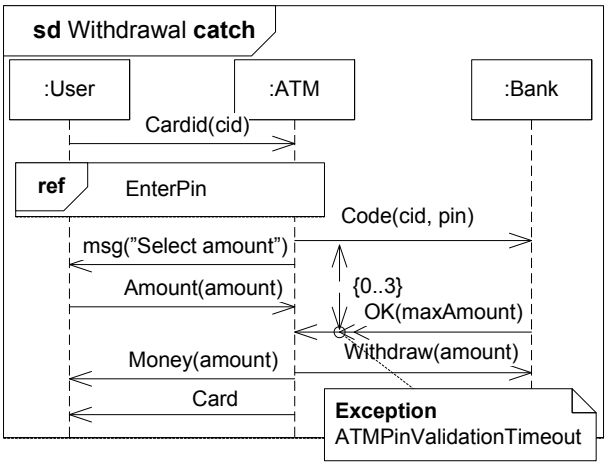

Fig. 3. Withdrawal with time exception

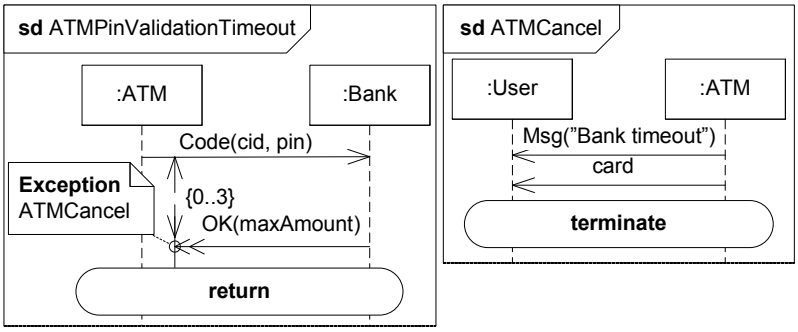

Fig. 4. Handling of pin validation timeout on the ATM

the events enabled but not executed at the trigger. Such enabled events may happen in parallel with the exception handling.

If we apply this to Withdrawal, Fig. 3, we notice that the ATM must at least send a code for validation to the bank before the timeout event may occur. Actually the exception may only occur more than three time units after the sending of the validation request. That is, before the ATMPinValidationTimeout may occur the user must have given a card, entered the pin, the ATM must have sent the pin for validation and three time units must have elapsed. After a possible recovery from the ATMPinValidationTimeout exception we can continue with sending the withdrawal message and returning the card and money.

The challenging part is how to handle the selection of amount if an exception occurs. Since these events are enabled they may happen in parallel with the exception. That is because the user is outside the ATMs sphere of control. We have three separate lifelines (User, ATM and Bank) that each communicates with the others through messages. Each lifeline in this distributed environment

is considered autonomous meaning that they are independent processes. We may therefore run the exception handling in parallel with other enabled events.

By enabled events we mean events that may happen regardless of whether the exception occurs or not. In the ATM example, an enabled event is the consumption of msg("Select amount"), and events only depending on that (here: user sending Amount). These events are outside the control of the exception handling, and must be allowed to continue. An example of a non-enabled event is the sending of Money from the ATM. This event can never be sent before the OK message is received.

4 The Formal Semantic Domain of Sequence Diagrams

In this section we briefly recount the main parts of the semantics of timed sequence diagrams as defined in [4]. In Sect. 5 we give our proposal for how this semantics may be extended to handle time exceptions.

Formally, we use denotational trace semantics in order to capture the meaning of sequence diagrams. A trace is a sequence of events, representing one run of the system. As explained in Sect. 2.3, we have three kinds of events: the sending, reception and consumption of a message, denoted by !, $\sim$ and ?, respectively. A message is a triple (s, tr, re) consisting of a signal s (the content of the message), a transmitter tr and a receiver re. The transmitter and receiver are lifelines, or possibly gates. (For a formal treatment of gates, see [5].)

Each event in the sequence diagram has a unique timestamp tag to which real timestamps will be assigned. Time constraints are expressed as logical formulas with these timestamp tags as free variables. Formally, an event is a triple (k, m, t) of a kind k (sending, reception or consumption), a message m and a timestamp tag t. As an example, EnterPin in Fig. 2 consists of six events: $(!, m, t_1)$, $(\sim, m, t_2)$, $(?, m, t_3)$, $(!, d, t_4)$, $(\sim, d, t_5)$ and $(?, d, t_6)$ where $m = (msg(Enterpin), ATM, User)$ and $d = (Digit, User, ATM)$. Notice that in Fig 2 the reception events are implicit, meaning that they may happen at any time between the corresponding send and receive events. The given time constraint may be written as $t_6 \leq t_1 + 5$.

$\mathcal{H}$ denotes the set of all well-formed traces. For a trace to be well-formed, it is required that

- for each message, the send event occurs before the receive event if both events are present in the trace.
- for each message, the receive event occurs before the consumption event if both events are present in the trace.
- the events in the trace are ordered by time.

$\mathcal{E}$ denotes the set of all syntactic events, and $[\![\mathcal{E}]\!]$ is the set of all corresponding semantical events with real timestamps assigned to the tags:

$$[\![\mathcal{E}]\!] \stackrel{\mathsf{def}}{=} \{(k, m, t \mapsto r) \mid (k, m, t) \in \mathcal{E} \wedge r \in \mathbb{R}\} \tag{1}$$

Parallel composition $s_1 \parallel s_2$ of two trace-sets is the set of all traces such that all events from one trace in s_1 and one trace in s_2 are included (and no other events), and the ordering of events from each of the traces is preserved. Formally:

$$s_1 \parallel s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists p \in \{1,2\}^\infty : \pi_2((\{1\} \times [\![\mathcal{E}]\!]) \circledcirc (p,h)) \in s_1 \ \wedge \ \pi_2((\{2\} \times [\![\mathcal{E}]\!]) \circledcirc (p,h)) \in s_2\} \quad (2)$$

The definition makes use of an oracle, the infinite sequence p, to determine the order in which the events from each trace are sequenced. π_2 is a projection operator returning the second element of a pair, and $\circledcirc$ is an operator filtering pairs of sequences with respect to pairs of elements.

Weak sequencing, $s_1 \gtrsim s_2$, is the set of all traces obtained by selecting one trace h_1 from s_1 and one trace h_2 from s_2 such that on each lifeline, the events from h_1 are ordered before the events from h_2:

$$s_1 \gtrsim s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : h \restriction l = h_1 \restriction l \frown h_2 \restriction l\} \quad (3)$$

where $\mathcal{L}$ is the set of all lifelines, $\frown$ is the concatenation operator on sequences, and $h \restriction l$ is the trace h with all events not taking place on the lifeline l removed.

Time constraint keeps only traces that are in accordance with the constraint:

$$s \wr C \stackrel{\text{def}}{=} \{h \in s \mid h \models C\} \quad (4)$$

where $h \models C$ holds if the timestamps in h does not violate C.

The semantics $[\![d]\!]$ of a sequence diagram d is given as a pair (p, n), where p is the set of positive and n the set of negative traces. Parallel composition, weak sequencing, time constraint and inner union ($\uplus$) of such pairs are defined as follows:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel (p_2 \cup n_2)) \cup (n_2 \parallel p_1)) \quad (5)$$

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \gtrsim p_2, (n_1 \gtrsim (n_2 \cup p_2)) \cup (p_1 \gtrsim n_2)) \quad (6)$$

$$(p, n) \wr C \stackrel{\text{def}}{=} (p \wr C, n \cup (p \wr \neg C)) \quad (7)$$

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \quad (8)$$

Finally, the semantics of the sequence diagram operators of interest in this paper are defined by:

$$[\![d_1 \ \mathsf{alt} \ d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \uplus [\![d_2]\!] \quad (9)$$

$$[\![d_1 \ \mathsf{par} \ d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \parallel [\![d_2]\!] \quad (10)$$

$$[\![d_1 \ \mathsf{seq} \ d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \gtrsim [\![d_2]\!] \quad (11)$$

$$[\![d \ \mathsf{tc} \ C]\!] \stackrel{\text{def}}{=} [\![d]\!] \wr C \quad (12)$$

$$[\![\mathsf{skip}]\!] \stackrel{\text{def}}{=} (\{\langle\rangle\}, \emptyset) \quad (13)$$

where tc is the operator used for time constraints and skip is the empty diagram (i.e. doing nothing). Definitions of other operators may be found in e.g. [5].

5 The Formal Semantics of Time Exceptions

In Sect. 3 we informally explained the semantics of time exceptions. In this section we define the semantics formally, based on the formalism introduced in Sect. 4. Furthermore we give theorems stating some desirable properties related to time exceptions and refinement. Due to lack of space, we have omitted the proofs from this paper. However, proofs may be found in [3].

5.1 Definitions

An exception diagram is mainly specified using the same operators as ordinary sequence diagrams, and its semantics may be calculated using the definitions given in Sect. 4. As explained in Sect. 3, the additional constructs used in exception diagrams is that the exception handling always ends with either return or terminate. Formally, the semantics of an exception (sub-)diagram marked with either return or terminate is defined by:

$$[\![\ d \ \mathsf{return}\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ d\ ]\!] \tag{14}$$

$$[\![\ d \ \mathsf{terminate}\ ]\!] \stackrel{\mathsf{def}}{=} appendTT([\![\ d\ ]\!]) \tag{15}$$

where $appendTT$ is a function appending a special termination event TT to every trace in its operand (i.e. all the positive and negative traces in $[\![\ d\ ]\!]$).

With this new termination event, weak sequencing of trace sets must be redefined so that traces that end with termination are not continued:

$$\begin{aligned} s_1 \gtrsim s_2 \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \\ (term(h_1) \wedge h = h_1) \ \vee \ (\neg term(h_1) \wedge \forall l \in \mathcal{L} : h \restriction l = h_1 \restriction l \frown h_2 \restriction l)\} \end{aligned} \tag{16}$$

where $term(h_1)$ is a boolean function that returns true if h_1 ends with the termination event TT, and false otherwise.

For parallel composition of trace sets, the traces may be calculated as before and then removing all events that occur after TT from the trace:

$$s_1 \parallel s_2 \stackrel{\mathsf{def}}{=} \{h \in \mathcal{H} \mid \exists h' \in s_1 \parallel' s_2 : h = chopTT(h')\} \tag{17}$$

where $\parallel'$ is parallel composition as defined by definition 2 and $chopTT$ is a function removing all events occurring after a potential TT in the trace.

A sequence diagram d marked as catching termination events then has the semantic effect that the termination mark is removed from the trace, meaning that the trace continues as specified by the diagram that is enclosing d:

$$[\![\ d \ \mathsf{catch}\ ]\!] \stackrel{\mathsf{def}}{=} removeTT([\![\ d\ ]\!]) \tag{18}$$

where $removeTT$ is a function removing TT from all traces in its operand.

Finally, we need to define the semantics of a sequence diagram which contains exceptions. The kind of exceptions considered in this paper is always connected

to a time constraint on an event. Syntactically, we write d tc $(C$ exception $e)$ to specify that d is a sequence diagram with time constraint C, and that the sequence diagram e specifies the exception handling in case C is violated. We use $q(C)$ to denote the event constrained by C, and $ll(C)$ to denote the lifeline on which this event occurs.

Obviously, a trace should be negative if the exception handling starts before the time constraint is actually violated. As an example, consider the specification of EnterPin in Fig. 2. Here, we have the constraint $t_6 \leq t_1 + 5$ as explained in Sect. 4. Letting t_7 be the timestamp of the sending of the message in UserLeftCard, we then intuitively have the corresponding constraint $t_7 > t_1 + 5$. Formally, we let e_C be the exception diagram where the time constraint C has been transformed into the corresponding time constraint for the first event in e (or several such constraints if there is a choice of first event for e).

The semantics of a sequence diagram with an exception is then defined by:

$$[\![\ d \text{ tc } (C \text{ exception } e)\ ]\!] \stackrel{\text{def}}{=} [\![\ d \text{ tc } C\ ]\!] \uplus \{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\ d[e_C/q(C)]\ ]\!] \restriction ll(C)\} \circledS [\![\ d[\mathsf{skip}/q(C)] \text{ par } e_C\ ]\!] \tag{19}$$

where $d[d_{new}/d_{old}]$ is the sequence diagram d with the sub-diagram d_{new} substituted for d_{old}, $\circledS$ is a filtering operator such that $S \circledS (p, n)$ is the pair (p, n) where all traces that are not in the set S are removed, $h \in (p, n)$ is a short-hand for $h \in p \vee h \in n$, and $\restriction$ is overloaded from traces to pairs of sets of traces in standard pointwise manner.

In definition 19, the first part corresponds to the semantics without the exception. The second part is all traces where the event $q(C)$ has not occurred, and the exception handling is performed instead. $[\![\ d[\mathsf{skip}/q(C)] \text{ par } e_C\ ]\!]$ gives the diagram d without the triggering event $q(C)$, executed in parallel with the exception e. However, this set is too comprehensive as we require that the lifeline of the triggering event, the lifeline $ll(C)$, must perform all of the exception handling before continuing with the original diagram. This is expressed by the set preceding the filtering operator.

5.2 Refinement

TimedSTAIRS [4] defines supplementing and narrowing as two special cases of refinement. Supplementing means adding more positive or negative traces to the sequence diagram, while narrowing means redefining earlier inconclusive traces as negative. Formally, a diagram d' with semantics (p', n') is said to be a refinement of another diagram d with semantics (p, n), written $d \rightsquigarrow d'$, iff

$$n \subseteq n' \ \wedge \ p \subseteq p' \cup n' \tag{20}$$

It should be clear from our explanations in Sect. 3 that adding exception handling to a sequence diagram constitutes a refinement. Adding a time constraint is an example of narrowing, as traces with invalid timestamps are moved from positive to negative when introducing the time constraint. More generally, we have the following theorem:

Theorem 1. *Assuming that the exception diagram e is not equivalent to the triggering event $q(C)$, i.e. $[\![e]\!] \neq (\{\langle q(C)\rangle\}, \emptyset)$, we have that*

1. $d \rightsquigarrow d \ \mathsf{tc}\ C$
2. $d \ \mathsf{tc}\ C \rightsquigarrow d \ \mathsf{tc}\ (C \ \mathsf{exception}\ e)$
3. $d \rightsquigarrow d \ \mathsf{tc}\ (C \ \mathsf{exception}\ e)$

Finally, the following theorem demonstrates that for a diagram containing exceptions, the normal and exceptional behavior may be refined separately:

Theorem 2. *Refinement is monotonic with respect to exceptions as defined by definition 19, i.e.:*

$$d \rightsquigarrow d \wedge e \rightsquigarrow e' \Rightarrow d \ \mathsf{tc}\ (C \ \mathsf{exception}\ e) \rightsquigarrow d' \ \mathsf{tc}\ (C \ \mathsf{exception}\ e')$$

6 Conclusions

We have shown that introducing time exceptions improve the completeness of sequence diagram descriptions while keeping the readability of the main specification. We have defined concrete notation for exceptions built on existing symbols of UML 2.1 and the simple time notation. Finally, we have given a precise formal definition of time exceptions and shown that our concepts are compositional since refinement is monotonic with respect to exceptions.

References

1. ETSI. *The Testing and Test Control Notation version 3 (TTCN-3); Part 1: TTCN-3 Core Language*, document: European Standard (ES) 201 873-1 version 2.2.1 (2003-02). Also published as ITU-T Recommendation Z.140 edition, 2003.
2. Oddleif Halvorsen and Øystein Haugen. Proposed notation for exception handling in UML 2 sequence diagrams. In *Australian Software Engineering Conference (ASWEC)*, pages 29–40. IEEE Computer Society, 2006.
3. Oddleif Halvorsen, Ragnhild Kobro Runde, and Øystein Haugen. Time exceptions in sequence diagrams. Technical Report 344, Department of Informatics, University of Oslo, 2006.
4. Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 1–25. Springer, 2005.
5. Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2006.
6. Object Management Group. *OMG Unified Modeling Language 1.4*, 2000.
7. Object Management Group. *UML profile for Schedulability, Performance and Time Specification*, document: ptc/05-01-02 edition, 2005.
8. Object Management Group. *UML Testing Profile*, document: ptc/05-07-07 edition, 2005.
9. Object Management Group. *UML 2.1 Superstructure Specification*, document: ptc/06-04-02 edition, 2006.

10. Ragnhild Kobro Runde, Atle Refsdal, and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. part 1: underspecification and inherent nondeterminism. Technical Report 346, Department of Informatics, University of Oslo, 2007.

A Proofs

A.1 Adding time constraints and time exceptions

Theorem 1 in Sect. 5.2 states that *assuming that the exception diagram e is not equivalent to the triggering event $q(C)$, i.e. $[\![e]\!] \neq (\{\langle q(C)\rangle\}, \emptyset)$, we have that*

1. $d \rightsquigarrow d \ \mathsf{tc}\ C$
2. $d \ \mathsf{tc}\ C \rightsquigarrow d \ \mathsf{tc}\ (C \ \mathsf{exception}\ e)$
3. $d \rightsquigarrow d \ \mathsf{tc}\ (C \ \mathsf{exception}\ e)$

1. PROOF SKETCH: Follows from definition 12 of time constraint, which ensures that all traces of the original diagram are also traces of the diagram with the time constraint added.
   PROOF:

   $\langle 1\rangle 1$. LET: $[\![d]\!] = (p, n)$
   $\langle 1\rangle 2$. $[\![d \ \mathsf{tc}\ C]\!] = (p \wr C, n \cup (p \wr \neg C))$
   $\langle 2\rangle 1$. $[\![d \ \mathsf{tc}\ C]\!] = [\![d]\!] \wr C$
   PROOF: Definition 12.
   $\langle 2\rangle 2$. $[\![d]\!] \wr C = (p \wr C, n \cup (p \wr \neg C))$
   PROOF: $\langle 1\rangle 1$ and definition 4.
   $\langle 2\rangle 3$. Q.E.D.
   PROOF: $\langle 2\rangle 1$ and $\langle 2\rangle 2$.
   $\langle 1\rangle 3$. $(p, n) \rightsquigarrow (p \wr C, n \cup (p \wr \neg C))$
   $\langle 2\rangle 1$. $n \subseteq n \cup (p \wr \neg C)$
   PROOF: Basic set theory.
   $\langle 2\rangle 2$. $p \subseteq (p \wr C) \cup (n \cup (p \wr \neg C))$
   $\langle 3\rangle 1$. $p \wr C = \{h \in p \mid h \models C\}$
   PROOF: Definition 4.
   $\langle 3\rangle 2$. $p \wr \neg C = \{h \in p \mid h \models \neg C\}$
   PROOF: Definition 4.
   $\langle 3\rangle 3$. $(p \wr C) \cup (p \wr \neg C) = p$
   PROOF: $\langle 3\rangle 1$, $\langle 3\rangle 2$ and basic set theory, as either $h \models C$ or $h \models \neg C$ for all traces h and time constraints C.
   $\langle 3\rangle 4$. Q.E.D.
   PROOF: $\langle 3\rangle 3$ and basic set theory.
   $\langle 2\rangle 3$. Q.E.D.
   PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$ and definition 20 of $\rightsquigarrow$.
   $\langle 1\rangle 4$. Q.E.D.
   PROOF: $\langle 1\rangle 1$, $\langle 1\rangle 2$ and $\langle 1\rangle 3$.

   □

2. PROOF SKETCH: Follows from definition 19 of exception, as the semantics of the original diagram is included as the first part of the definition.
   PROOF:

   $\langle 1\rangle 1$. LET: $(p_1, n_1) = [\![d \ \mathsf{tc}\ C]\!]$

$\langle 1\rangle 2$. LET: $(p_2, n_2) =$
$\{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\, d[e_C/q(C)] \,]\!] \restriction ll(C)\} \circledS [\![\, d[\mathsf{skip}/q(C)] \text{ par } e_C \,]\!]$
$\langle 1\rangle 3$. $[\![\, d \text{ tc } (C \text{ exception } e) \,]\!] = (p_1 \cup p_2, n_1 \cup n_2)$
$\langle 2\rangle 1$. $[\![\, d \text{ tc } (C \text{ exception } e) \,]\!] = (p_1, n_1) \uplus (p_2, n_2)$
PROOF: $\langle 1\rangle 1$, $\langle 1\rangle 2$ and definition 19 of exception.
$\langle 2\rangle 2$. $(p_1, n_1) \uplus (p_2, n_2) = (p_1 \cup p_2, n_1 \cup n_2)$
PROOF: Definition 8 of $\uplus$.
$\langle 2\rangle 3$. Q.E.D.
PROOF: $\langle 2\rangle 1$ and $\langle 2\rangle 2$.
$\langle 1\rangle 4$. $(p_1, n_1) \rightsquigarrow (p_1 \cup p_2, n_1 \cup n_2)$
$\langle 2\rangle 1$. $n_1 \subseteq n_1 \cup n_2$
PROOF: Basic set theory.
$\langle 2\rangle 2$. $p_1 \subseteq (p_1 \cup p_2) \cup (n_1 \cup n_2)$
PROOF: Basic set theory.
$\langle 2\rangle 3$. Q.E.D.
PROOF: $\langle 2\rangle 1$, $\langle 2\rangle 2$ and definition 20 of $\rightsquigarrow$.
$\langle 1\rangle 5$. Q.E.D.
PROOF: $\langle 1\rangle 1$, $\langle 1\rangle 3$ and $\langle 1\rangle 4$.

□

3. PROOF: Follows directly from the two previous facts using that refinement is transitive (lemma 26 in [5]).

□

A.2 Monotonicity

First, we formally define *appendTT* and *removeTT*:

$$appendTT((p, n)) \stackrel{\text{def}}{=} (appendTT(p), appendTT(n)) \tag{21}$$

$$removeTT((p, n)) \stackrel{\text{def}}{=} (removeTT(p), removeTT(n)) \tag{22}$$

where *appendTT* and *removeTT* on trace-sets are defined by:

$$appendTT(s) \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h' \in s : h = h' \frown \langle TT \rangle\} \tag{23}$$

$$removeTT(s) \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h' \in s : h = h' \setminus \{TT\}\} \tag{24}$$

where $\setminus$ is a hiding operator such that $t \setminus A$ removes all events in the set A from the trace t.

Lemma 1. $s \subseteq s' \Rightarrow appendTT(s) \subseteq appendTT(s')$

PROOF:

$\langle 1\rangle 1$. ASSUME: $s \subseteq s'$
PROVE: $appendTT(s) \subseteq appendTT(s')$
$\langle 2\rangle 1$. CASE: $appendTT(s) = \emptyset$

PROOF: Trivial, as $\emptyset \subseteq A$ for all sets A.
⟨2⟩2. CASE: $appendTT(s) \neq \emptyset$
⟨3⟩1. Choose arbitrary $t \in appendTT(s)$
PROOF: $appendTT(s)$ is non-empty by ⟨2⟩2.
⟨3⟩2. Choose $t' \in s$ such that $t = t' \frown \langle TT \rangle$
PROOF: ⟨3⟩1 and definition 23 of $appendTT$.
⟨3⟩3. $t' \in s'$
PROOF: ⟨3⟩2 and ⟨1⟩1.
⟨3⟩4. $t \in appendTT(s')$
PROOF: ⟨3⟩2, ⟨3⟩3 and definition 23 of $appendTT$.
⟨3⟩5. Q.E.D.
PROOF: ⟨3⟩1, ⟨3⟩4 and definition of $\subseteq$.
⟨2⟩3. Q.E.D.
PROOF: The cases are exhaustive.
⟨1⟩2. Q.E.D.
PROOF: $\Rightarrow$-rule.

□

Lemma 2. $s \subseteq s' \Rightarrow removeTT(s) \subseteq removeTT(s')$

PROOF:

⟨1⟩1. ASSUME: $s \subseteq s'$
PROVE: $removeTT(s) \subseteq removeTT(s')$
⟨2⟩1. CASE: $removeTT(s) = \emptyset$
PROOF: Trivial, as $\emptyset \subseteq A$ for all sets A.
⟨2⟩2. CASE: $removeTT(s) \neq \emptyset$
⟨3⟩1. Choose arbitrary $t \in removeTT(s)$
PROOF: $removeTT(s)$ is non-empty by ⟨2⟩2.
⟨3⟩2. Choose $t' \in s$ such that $t = t' \setminus \{TT\}$
PROOF: ⟨3⟩1 and definition 24 of $removeTT$.
⟨3⟩3. $t' \in s'$
PROOF: ⟨3⟩2 and ⟨1⟩1.
⟨3⟩4. $t \in removeTT(s')$
PROOF: ⟨3⟩2, ⟨3⟩3 and definition 24 of $removeTT$.
⟨3⟩5. Q.E.D.
PROOF: ⟨3⟩1, ⟨3⟩4 and definition of $\subseteq$.
⟨2⟩3. Q.E.D.
PROOF: The cases are exhaustive.
⟨1⟩2. Q.E.D.
PROOF: $\Rightarrow$-rule.

□

Lemma 3. $s_1 \subseteq s_1' \wedge s_2 \subseteq s_2' \Rightarrow s_1 \succsim s_2 \subseteq s_1' \succsim s_2'$

PROOF:

⟨1⟩1. ASSUME: 1. $s_1 \subseteq s_1'$

2. $s_2 \subseteq s_2'$

PROVE: $s_1 \succsim s_2 \subseteq s_1' \succsim s_2'$

$\langle 2 \rangle 1$. Choose arbitrary $h \in s_1 \succsim s_2$

PROOF: The lemma is trivially true if $s_1 \succsim s_2$ is empty.

$\langle 2 \rangle 2$. $h \in s_1' \succsim s_2'$

$\langle 3 \rangle 1$. CASE: Choose $h_1 \in s_1$ such that $term(h_1)$ and $h = h_1$

$\langle 4 \rangle 1$. $h_1 \in s_1'$

PROOF: $\langle 3 \rangle 1$ and $\langle 1 \rangle 1{:}1$.

$\langle 4 \rangle 2$. $h_1 \in s_1' \succsim s_2'$

PROOF: $\langle 3 \rangle 1$, $\langle 4 \rangle 1$ and definition 16 of $\succsim$.

$\langle 4 \rangle 3$. Q.E.D.

PROOF: $\langle 3 \rangle 1$ and $\langle 4 \rangle 2$.

$\langle 3 \rangle 2$. CASE: Choose $h_1 \in s_1$ and $h_2 \in s_2$ such that $\neg term(h_1)$ and $\forall l \in \mathcal{L} : h \restriction l = h_1 \restriction l \frown h_2 \restriction l$

$\langle 4 \rangle 1$. $h_1 \in s_1'$

PROOF: $\langle 3 \rangle 2$ and $\langle 1 \rangle 1{:}1$.

$\langle 4 \rangle 2$. $h_2 \in s_2'$

PROOF: $\langle 3 \rangle 2$ and $\langle 1 \rangle 1{:}2$.

$\langle 4 \rangle 3$. $h \in s_1' \succsim s_2'$

PROOF: $\langle 3 \rangle 2$, $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ and definition 16 of $\succsim$.

$\langle 4 \rangle 4$. Q.E.D.

$\langle 3 \rangle 3$. Q.E.D.

PROOF: The cases are exhaustive by definition 16 of $\succsim$.

$\langle 2 \rangle 3$. Q.E.D.

PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and definition of $\subseteq$.

$\langle 1 \rangle 2$. Q.E.D.

PROOF: $\Rightarrow$-rule.

□

Lemma 4. $s_1 \subseteq s_1' \wedge s_2 \subseteq s_2' \Rightarrow s_1 \parallel s_2 \subseteq s_1' \parallel s_2'$

PROOF:

$\langle 1 \rangle 1$. ASSUME: 1. $s_1 \subseteq s_1'$

2. $s_2 \subseteq s_2'$

PROVE: $s_1 \parallel s_2 \subseteq s_1' \parallel s_2'$

$\langle 2 \rangle 1$. Choose arbitrary $h \in s_1 \parallel s_2$

PROOF: The lemma is trivially true if $s_1 \parallel s_2$ is empty.

$\langle 2 \rangle 2$. $h \in s_1' \parallel s_2'$

$\langle 3 \rangle 1$. Choose $h' \in s_1 \parallel' s_2$ such that $h = chopTT(h')$

PROOF: $\langle 2 \rangle 1$ and definition 17 of $\parallel$.

$\langle 3 \rangle 2$. $h' \in s_1' \parallel' s_2'$

PROOF: $\langle 1 \rangle 1{:}1$, $\langle 1 \rangle 1{:}2$, $\langle 3 \rangle 1$ and lemma 28 in [5].

$\langle 3 \rangle 3$. $h \in s_1' \parallel s_2'$

PROOF: $\langle 3 \rangle 1$, $\langle 3 \rangle 2$ and definition 17 of $\parallel$.

$\langle 3 \rangle 4$. Q.E.D.

$\langle 2 \rangle 3$. Q.E.D.

PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and definition of $\subseteq$.
$\langle 1 \rangle 2$. Q.E.D.
PROOF: $\Rightarrow$-rule.

□

Theorem 3. *The refinement relation $\rightsquigarrow$ is monotonic with respect to* seq*, i.e.*

$$d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \Rightarrow (d_1 \text{ seq } d_2) \rightsquigarrow (d_1' \text{ seq } d_2')$$

PROOF: Lemma 30 in [5], using lemma 3 instead of lemma 27 in [5].

□

Theorem 4. *The refinement relation $\rightsquigarrow$ is monotonic with respect to* par*, i.e.*

$$d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \Rightarrow (d_1 \text{ par } d_2) \rightsquigarrow (d_1' \text{ par } d_2')$$

PROOF: Lemma 31 in [5], using lemma 4 instead of lemma 28 in [5].

□

Theorem 5. *The refinement relation $\rightsquigarrow$ is monotonic with respect to* return*, i.e.*

$$d \rightsquigarrow d' \Rightarrow (d \text{ return}) \rightsquigarrow (d' \text{ return})$$

PROOF:

$\langle 1 \rangle 1$. ASSUME: $d \rightsquigarrow d'$
PROVE: $(d \text{ return}) \rightsquigarrow (d' \text{ return})$
$\langle 2 \rangle 1$. $[\![d \text{ return}]\!] = [\![d]\!]$
PROOF: Definition 14.
$\langle 2 \rangle 2$. $[\![d' \text{ return}]\!] = [\![d']\!]$
PROOF: Definition 14.
$\langle 2 \rangle 3$. Q.E.D.
PROOF: $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ and $\langle 1 \rangle 1$.
$\langle 1 \rangle 2$. Q.E.D.
PROOF: $\Rightarrow$-rule.

□

Theorem 6. *The refinement relation $\rightsquigarrow$ is monotonic with respect to* terminate*, i.e.*

$$d \rightsquigarrow d' \Rightarrow (d \text{ terminate}) \rightsquigarrow (d' \text{ terminate})$$

PROOF:

$\langle 1 \rangle 1$. LET: $[\![d]\!] = (p, n)$
$\langle 1 \rangle 2$. LET: $[\![d']\!] = (p', n')$
$\langle 1 \rangle 3$. ASSUME: $d \rightsquigarrow d'$
PROVE: $(d \text{ terminate}) \rightsquigarrow (d' \text{ terminate})$
$\langle 2 \rangle 1$. $[\![d \text{ terminate}]\!] = (appendTT(p), appendTT(n))$
PROOF: $\langle 1 \rangle 1$, definition 15 of terminate and definitions 21 and 23 of $appendTT$.

⟨2⟩2. $[\![\ d'\ \mathsf{terminate}\ ]\!] = (appendTT(p'), appendTT(n'))$
PROOF: ⟨1⟩2, definition 15 of terminate and definitions 21 and 23 of $appendTT$.
⟨2⟩3. $(appendTT(p), appendTT(n)) \rightsquigarrow (appendTT(p'), appendTT(n'))$
⟨3⟩1. $appendTT(n) \subseteq appendTT(n')$
⟨4⟩1. $n \subseteq n'$
PROOF: ⟨1⟩3 and definition 20 of $\rightsquigarrow$.
⟨4⟩2. Q.E.D.
PROOF: ⟨4⟩1 and lemma 1.
⟨3⟩2. $appendTT(p) \subseteq appendTT(p') \cup appendTT(n')$
⟨4⟩1. $p \subseteq p' \cup n'$
PROOF: ⟨1⟩3 and definition 20 of $\rightsquigarrow$.
⟨4⟩2. $appendTT(p) \subseteq appendTT(p' \cup n')$
PROOF: ⟨4⟩1 and lemma 1.
⟨4⟩3. $appendTT(p) \subseteq appendTT(p') \cup appendTT(n')$
PROOF: ⟨4⟩2 and definition 23 of $appendTT$.
⟨4⟩4. Q.E.D.
⟨3⟩3. Q.E.D.
PROOF: ⟨3⟩1, ⟨3⟩2 and definition 20 of $\rightsquigarrow$.
⟨2⟩4. Q.E.D.
PROOF: ⟨2⟩1, ⟨2⟩2 and ⟨2⟩3.
⟨1⟩4. Q.E.D.
PROOF: ⟨1⟩3 and ⇒-rule.

□

Theorem 7. *The refinement relation $\rightsquigarrow$ is monotonic with respect to* catch*, i.e.*

$$d \rightsquigarrow d' \Rightarrow (d\ \mathsf{catch}) \rightsquigarrow (d'\ \mathsf{catch})$$

PROOF:

⟨1⟩1. $[\![\ d\ ]\!] = (p, n)$
⟨1⟩2. $[\![\ d'\ ]\!] = (p', n')$
⟨1⟩3. ASSUME: $d \rightsquigarrow d'$
PROVE: $(d\ \mathsf{catch}) \rightsquigarrow (d'\ \mathsf{catch})$
⟨2⟩1. $[\![\ d\ \mathsf{catch}\ ]\!] = (removeTT(p), removeTT(n))$
PROOF: ⟨1⟩1, definition 18 of catch and definitions 22 and 24 of $removeTT$.
⟨2⟩2. $[\![\ d'\ \mathsf{catch}\ ]\!] = (removeTT(p'), removeTT(n'))$
PROOF: ⟨1⟩2, definition 18 of catch and definitions 22 and 24 of $removeTT$.
⟨2⟩3. $(removeTT(p), removeTT(n)) \rightsquigarrow (removeTT(p'), removeTT(n'))$
⟨3⟩1. $removeTT(n) \subseteq removeTT(n')$
⟨4⟩1. $n \subseteq n'$
PROOF: ⟨1⟩3 and definition 20 of $\rightsquigarrow$.
⟨4⟩2. Q.E.D.
PROOF: ⟨4⟩1 and lemma 2.
⟨3⟩2. $removeTT(p) \subseteq removeTT(p') \cup removeTT(n')$
⟨4⟩1. $p \subseteq p' \cup n'$
PROOF: ⟨1⟩3 and definition 20 of $\rightsquigarrow$.

⟨4⟩2. $removeTT(p) \subseteq removeTT(p' \cup n')$
PROOF: ⟨4⟩1 and lemma 2.
⟨4⟩3. $removeTT(p) \subseteq removeTT(p') \cup removeTT(n')$
PROOF: ⟨4⟩2 and definition 24 of $removeTT$.
⟨4⟩4. Q.E.D.
⟨3⟩3. Q.E.D.
PROOF: ⟨3⟩1, ⟨3⟩2 and definition 20 of $\rightsquigarrow$.
⟨2⟩4. Q.E.D.
PROOF: ⟨2⟩1, ⟨2⟩2 and ⟨2⟩3.
⟨1⟩4. Q.E.D.
PROOF: ⟨1⟩3 and $\Rightarrow$-rule.

□

Lemma 5.

$$S \subseteq S' \wedge d \rightsquigarrow d' \Rightarrow S \circledS [\![d]\!] \rightsquigarrow S' \circledS [\![d']\!]$$

PROOF:

⟨1⟩1. LET: $[\![d]\!] = (p, n)$
⟨1⟩2. LET: $[\![d']\!] = (p', n')$
⟨1⟩3. ASSUME: 1. $S \subseteq S'$
2. $d \rightsquigarrow d'$
PROVE: $S \circledS [\![d]\!] \rightsquigarrow S' \circledS [\![d']\!]$
⟨2⟩1. $(S \circledS p, S \circledS n) \rightsquigarrow (S' \circledS p', S' \circledS n')$
⟨3⟩1. $S \circledS n \subseteq S' \circledS n'$
⟨4⟩1. Choose arbitrary $t \in S \circledS n$
PROOF: ⟨3⟩1 is trivially true if $S \circledS n = \emptyset$.
⟨4⟩2. $t \in S \wedge t \in n$
PROOF: ⟨4⟩1 and definition of $\circledS$.
⟨4⟩3. $t \in S'$
PROOF: ⟨4⟩2 and ⟨1⟩3:1.
⟨4⟩4. $t \in n'$
PROOF: ⟨4⟩2, ⟨1⟩3:2 and definition 20 of $\rightsquigarrow$.
⟨4⟩5. $t \in S' \circledS n'$
PROOF: ⟨4⟩3, ⟨4⟩4 and definition of $\circledS$.
⟨4⟩6. Q.E.D.
PROOF: ⟨4⟩1, ⟨4⟩5 and definition of $\subseteq$.
⟨3⟩2. $S \circledS p \subseteq (S' \circledS p') \cup (S \circledS n')$
⟨4⟩1. Choose arbitrary $t \in S \circledS p$
PROOF: ⟨3⟩1 is trivially true if $S \circledS p = \emptyset$.
⟨4⟩2. $t \in S \wedge t \in p$
PROOF: ⟨4⟩1 and definition of $\circledS$.
⟨4⟩3. $t \in S'$
PROOF: ⟨4⟩2 and ⟨1⟩3:1.
⟨4⟩4. $t \in p' \cup n'$
PROOF: ⟨4⟩2, ⟨1⟩3:2 and definition 20 of $\rightsquigarrow$.
⟨4⟩5. $t \in S' \circledS (p' \cup n')$

PROOF: $\langle 4\rangle 3$, $\langle 4\rangle 4$ and definition of $\circledS$.

$\langle 4\rangle 6$. $t \in (S' \circledS p') \cup (S' \circledS n')$

PROOF: $\langle 4\rangle 5$ and definition of $\circledS$.

$\langle 4\rangle 7$. Q.E.D.

PROOF: $\langle 4\rangle 1$, $\langle 4\rangle 6$ and definition of $\subseteq$.

$\langle 3\rangle 3$. Q.E.D.

PROOF: $\langle 3\rangle 1$, $\langle 3\rangle 2$ and definition 20 of $\rightsquigarrow$.

$\langle 2\rangle 2$. Q.E.D.

PROOF: $\langle 1\rangle 1$, $\langle 1\rangle 2$ and defintion of $\circledS$.

$\langle 1\rangle 4$. Q.E.D.

PROOF: $\langle 1\rangle 3$ and $\Rightarrow$-rule.

□

Theorem 8. *Refinement is monotonic with resepct to exceptions as defined by definition 19, i.e.:*

$$d \rightsquigarrow d' \wedge e \rightsquigarrow e' \Rightarrow d \ \mathsf{tc}\ (C\ \mathsf{exception}\ e) \rightsquigarrow d' \ \mathsf{tc}\ (C\ \mathsf{exception}\ e')$$

PROOF:

$\langle 1\rangle 1$. ASSUME: 1. $d \rightsquigarrow d'$

2. $e \rightsquigarrow e'$

PROVE: $d \ \mathsf{tc}\ (C\ \mathsf{exception}\ e) \rightsquigarrow d' \ \mathsf{tc}\ (C\ \mathsf{exception}\ e')$

$\langle 2\rangle 1$. $d \ \mathsf{tc}\ C \rightsquigarrow d' \ \mathsf{tc}\ C$

PROOF: $\langle 1\rangle 1$:1 and lemma 32 in [5] (monotonicity of $\rightsquigarrow$ with respect to tc).

$\langle 2\rangle 2$. $e_C \rightsquigarrow e'_C$

PROOF: $\langle 1\rangle 1$:2, definition of e_C (e with additional time constraints) and lemma 32 in [5] (monotonicity of $\rightsquigarrow$ with respect to tc).

$\langle 2\rangle 3$. $\{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\ d[e_C/q(C)]\ ]\!] \restriction ll(C)\} \subseteq$
$\{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\ d'[e'_C/q(C)]\ ]\!] \restriction ll(C)\}$

$\langle 3\rangle 1$. $\pi_1([\![\ d[e_C/q(C)]\ ]\!]) \cup \pi_2([\![\ d[e_C/q(C)]\ ]\!]) \subseteq \pi_1([\![\ d'[e'_C/q(C)]\ ]\!]) \cup \pi_2([\![\ d'[e'_C/q(C)]\ ]\!])$

$\langle 4\rangle 1$. $d[e_C/q(C)] \rightsquigarrow d'[e'_C/q(C)]$

PROOF: $\langle 1\rangle 1$:1, $\langle 2\rangle 2$ and monotonicity of $\rightsquigarrow$ with respect to all of the operators used in d (Theorems 3–7 together with the monotonicity theorems in [5]).

$\langle 4\rangle 2$. $\pi_2([\![\ d[e_C/q(C)]\ ]\!]) \subseteq \pi_2([\![\ d'[e'_C/q(C)]\ ]\!])$

PROOF: $\langle 4\rangle 1$ and definition 20 of $\rightsquigarrow$.

$\langle 4\rangle 3$. $\pi_1([\![\ d[e_C/q(C)]\ ]\!]) \subseteq \pi_1([\![\ d'[e'_C/q(C)]\ ]\!]) \cup \pi_2([\![\ d'[e'_C/q(C)]\ ]\!])$

PROOF: $\langle 4\rangle 1$ and definition 20 of $\rightsquigarrow$.

$\langle 4\rangle 4$. Q.E.D.

PROOF: $\langle 4\rangle 2$ and $\langle 4\rangle 3$.

$\langle 3\rangle 2$. Q.E.D.

PROOF: $\langle 3\rangle 1$, definition of $\restriction$ and basic set theory.

$\langle 2\rangle 4$. $d[\mathsf{skip}/q(C)]\ \mathsf{par}\ e_C \rightsquigarrow d'[\mathsf{skip}/q(C)]\ \mathsf{par}\ e'_C$

$\langle 3\rangle 1$. $d[\mathsf{skip}/q(C)] \rightsquigarrow d'[\mathsf{skip}/q(C)]$

PROOF: $\langle 1\rangle 1$:1 and monotonicity of $\rightsquigarrow$ with respect to all of the operators used in d (Theorems 3–7 together with the monotonicity theorems in [5]).

⟨3⟩2. Q.E.D.

PROOF: ⟨2⟩2, ⟨3⟩1 and Theorem 4 (monotonicity of $\rightsquigarrow$ with respect to par).

⟨2⟩5. $\{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\ d[e_C/q(C)]\ ]\!] \restriction ll(C)\} \circledS [\![\ d[\mathsf{skip}/q(C)]\ \mathsf{par}\ e_C\ ]\!] \rightsquigarrow \{h \in \mathcal{H} \mid h \restriction ll(C) \in [\![\ d'[e'_C/q(C)]\ ]\!] \restriction ll(C)\} \circledS [\![\ d'[\mathsf{skip}/q(C)]\ \mathsf{par}\ e'_C\ ]\!]$

PROOF: ⟨2⟩3, ⟨2⟩4 and lemma 5.

⟨2⟩6. Q.E.D.

PROOF: ⟨2⟩1, ⟨2⟩5, definition 19 of exception and Theorem 11 in [10] (monotonicity of $\rightsquigarrow$ with respect to $\uplus$).

⟨1⟩2. Q.E.D.

PROOF: $\Rightarrow$-rule.

□